

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of : THE COMMISSIONER IS AUTHORIZED
Taketo HEISHI et al. : TO CHARGE ANY DEFICIENCY IN THE
Serial No. NEW : FEES FOR THIS PAPER TO DEPOSIT
Filed July 31, 2003 : ACCOUNT NO. 23-0975
: Attn: APPLICATION BRANCH
: Attorney Docket No. 2003_1076A

COMPILER, COMPILER APPARATUS
AND COMPILATION METHOD

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

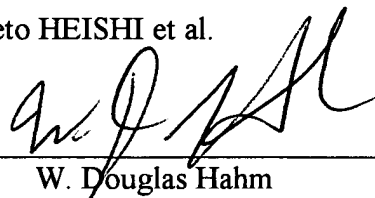
Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2002-226682, filed August 2, 2002, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Taketo HEISHI et al.

By



W. Douglas Hahm
Registration No. 44,142 *for*
Michael S. Huppert
Registration No. 40,268
Attorney for Applicants

WDH/kjf
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
July 31, 2003

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application:

2002年 8月 2日

出願番号
Application Number:

特願2002-226682

[ST.10/C]:

[JP2002-226682]

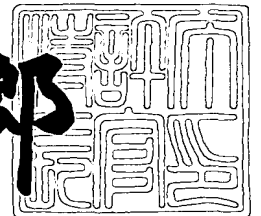
出願人
Applicant(s):

松下電器産業株式会社

2003年 2月28日

特許庁長官
Commissioner,
Japan Patent Office

太田信一郎



出証番号 出証特2003-3011925

【書類名】 特許願

【整理番号】 5037740106

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/45

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 瓶子 岳人

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 坂田 俊幸

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 小川 一

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 宮地 涼子

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 宮阪 修二

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 石川 智一

【特許出願人】

【識別番号】 000005821

【氏名又は名称】 松下電器産業株式会社

【代理人】

【識別番号】 100109210

【弁理士】

【氏名又は名称】 新居 広守

【電話番号】 06-4806-7530

【手数料の表示】

【予納台帳番号】 049515

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンパイラ用プログラム、コンパイラ装置及びコンパイル方法

【特許請求の範囲】

【請求項 1】 ソースプログラムを機械語プログラムに翻訳するコンパイラ用のプログラムであって、

対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、

前記プログラムは、

前記ソースプログラムを解析するパーサーステップと、

解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、

変換された中間コードを最適化する最適化ステップと、

最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、

前記中間コード変換ステップは、

前記中間コードのうち、前記演算処理定義情報で定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、

あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、

前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う

ことを特徴とするプログラム。

【請求項 2】 前記演算処理定義情報は、前記ソースプログラムにインクルードされるヘッダファイルであり、

前記ヘッダファイルには、データとメソッドとからなるクラスによって前記演算処理が定義され、

前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義されたクラスを参照しているものがあるか否かを検出することによって、前記演算処理を参照するしているものがあるか否かを検出する

ことを特徴とする請求項 1 記載のプログラム。

【請求項 3】 前記クラスは、固定小数点型を定義するクラスであり、
前記検出サブステップでは、前記固定小数点型のデータを使用している中間コードを検出する

ことを特徴とする請求項 2 記載のプログラム。

【請求項 4】 前記クラスのメソッドは、前記固定小数点型のデータを対象とする演算子を定義するメソッドであり、

前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、

前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換える

ことを特徴とする請求項 3 記載のプログラム。

【請求項 5】 前記クラスは、S I M D 型を定義するクラスであり、
前記検出サブステップでは、前記 S I M D 型のデータを使用している中間コードを検出する

ことを特徴とする請求項 2 記載のプログラム。

【請求項 6】 前記クラスのメソッドは、前記 S I M D 型のデータを対象とする演算子を定義するメソッドであり、

前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、

前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換える

ことを特徴とする請求項 5 記載のプログラム。

【請求項 7】 前記クラスには、対応する処理を実現する 1 つの機械語命令が対応づけられ、

前記置換サブステップでは、前記中間コードを、前記クラスに対応づけられた 1 つの機械語命令に置き換える

ことを特徴とする請求項 2 ～ 6 のいずれか 1 項に記載のプログラム。

【請求項 8】 前記クラスには、対応する処理を実現する 2 以上の機械語命

令が対応づけられ、

前記置換サブステップでは、前記中間コードを、前記クラスに対応づけられた 2 以上の機械語命令に置き換える

ことを特徴とする請求項 2 ～ 6 のいずれか 1 項に記載のプログラム。

【請求項 9】 前記演算処理定義情報は、前記ソースプログラムにインクルードされるヘッダファイルであり、

前記ヘッダファイルには、関数によって前記演算処理が定義され、

前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義された関数を参照しているものがあるか否かを検出することによって、前記演算処理を参照するしているものがあるか否かを検出する

ことを特徴とする請求項 1 記載のプログラム。

【請求項 10】 前記関数は、対応する処理を実現する 1 つの機械語命令が記述された関数であり、

前記置換サブステップでは、前記中間コードを前記関数に記述された 1 つの機械語命令に置き換える

ことを特徴とする請求項 9 記載のプログラム。

【請求項 11】 前記関数には、入力データの最上位ビットから 0 が連続するビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値について、最上位ビットから 0 が連続するビット数を数え、その結果を第 2 レジスタに格納する機械語命令である

ことを特徴とする請求項 10 記載のプログラム。

【請求項 12】 前記関数には、入力データの最上位ビットから 1 が連続するビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値について、最上位ビットから 1 が連続するビット数を数え、その結果を第 2 レジスタに格納する機械語命令である

ことを特徴とする請求項 10 記載のプログラム。

【請求項 13】 前記関数には、入力データの最上位ビットと同じ値が連続

するビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値について、最上位ビットと同じ値が連続するビット数を数え、その結果を第 2 レジスタに格納する機械語命令である

ことを特徴とする請求項 1 0 記載のプログラム。

【請求項 1 4】 前記関数は、入力データの最上位ビットと同じ値が最上位ビットの次から何ビット連続するかを返す関数であり、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値について、最上位ビットと同じ値が最上位ビットの次から何ビット連続するかを数え、その結果を第 2 レジスタに格納する機械語命令である

ことを特徴とする請求項 1 3 記載のプログラム。

【請求項 1 5】 前記関数には、入力データ中に含まれる 1 のビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値について、1 のビット数を数え、その結果を第 2 レジスタに格納する機械語命令であることを特徴とする請求項 1 0 記載のプログラム。

【請求項 1 6】 前記関数には、入力データから、指定されたビット位置のビットを抽出し、符号拡張した値を返す関数が含まれ、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値から、第 2 レジスタで指定されたビット位置のビットを取り出し、符号拡張して第 3 レジスタに格納する機械語命令である

ことを特徴とする請求項 1 0 記載のプログラム。

【請求項 1 7】 前記関数には、入力データから、指定されたビット位置のビットを抽出し、ゼロ拡張した値を返す関数が含まれ、

前記関数に記述されている機械語命令は、第 1 レジスタに格納された値から、第 2 レジスタで指定されたビット位置のビットを取り出し、ゼロ拡張して第 3 レジスタに格納する機械語命令である

ことを特徴とする請求項 1 0 記載のプログラム。

【請求項 1 8】 前記関数は、対応する処理を実現する 2 以上の機械語命令

からなる機械語命令列が記述された関数であり、

前記置換サブステップでは、前記中間コードを前記関数に記述された機械語命令列に置き換える

ことを特徴とする請求項 9 記載のプログラム。

【請求項 1 9】 前記関数には、モジュロアドレッシングのアドレス更新をする関数が含まれる

ことを特徴とする請求項 1 8 記載のプログラム。

【請求項 2 0】 前記関数に記述されている機械語命令列には、第 1 レジスタに格納された値の所定のビットフィールドを第 2 レジスタに格納された値で置き換えることによって得られる値を第 3 レジスタに格納する機械語命令が含まれる

ことを特徴とする請求項 1 9 記載のプログラム。

【請求項 2 1】 前記関数には、ビットリバースアドレッシングのアドレス更新をする関数が含まれる

ことを特徴とする請求項 1 8 記載のプログラム。

【請求項 2 2】 前記関数に記述されている機械語命令列には、第 1 レジスタに格納された値の所定のビットフィールドをビット単位で位置反転することによって得られる値を第 3 レジスタに格納する機械語命令が含まれる

ことを特徴とする請求項 2 1 記載のプログラム。

【請求項 2 3】 前記関数には、最適化におけるレジスタ割り付けの対象となる汎用レジスタに加えて、レジスタ割り付けの対象とならないアキュムレータも更新する演算であって、アキュムレータを参照型として一時変数を指定することが可能な関数が含まれる

ことを特徴とする請求項 9 記載のプログラム。

【請求項 2 4】 前記関数は、乗算を行う関数であって、乗算結果を格納するアキュムレータを参照型として一時変数を指定することが可能な関数である

ことを特徴とする請求項 2 3 記載のプログラム。

【請求項 2 5】 前記関数は、乗算を行う関数であって、乗算結果を格納するアキュムレータを参照型として一時変数を指定することが可能な関数である

ことを特徴とする請求項 2 3 記載のプログラム。

【請求項 2 6】 前記置換サブステップでは、前記関数を参照している中間コードを、当該関数の引数の種類に対応した種類のオペランドを持つ機械語命令に置き換える

ことを特徴とする請求項 9 記載のプログラム。

【請求項 2 7】 前記置換サブステップでは、
前記関数関数の引数がすべて定数である場合には、それらの定数を畳み込んで得られる定数値をオペランドを持つ機械語命令に置き換え、

前記関数関数の引数の一部が定数である場合には、即値オペランドを持つ機械語命令に置き換え、

前記関数関数の引数がすべて変数である場合には、レジスタオペランドを持つ機械語命令に置き換える

ことを特徴とする請求項 2 6 記載のプログラム。

【請求項 2 8】 前記最適化ステップは、異なる型間の演算を行う複数の中間コード又は複数の機械語命令を、当該演算を行う 1 つの機械語命令に置き換える型変換最適化サブステップを含む

ことを特徴とする請求項 1 記載のプログラム。

【請求項 2 9】 前記型変換サブステップでは、 n ビットの 2 つの変数を乗算し、結果を $2n$ ビットの変数に格納する演算を行う複数の中間コード又は複数の機械語命令を、当該演算を行う 1 つの機械語命令に置き換える

ことを特徴とする請求項 2 8 記載のプログラム。

【請求項 3 0】 前記型変換サブステップでは、前記 2 つの変数に対して、 n ビットから $2n$ ビットに型変換する明示的な宣言がなされている場合に、前記演算を前記機械語命令に置き換える

ことを特徴とする請求項 2 9 記載のプログラム。

【請求項 3 1】 前記コンパイラは、2 以上の固定小数点型を対象として演算する 2 以上の固定小数点モードを備えるプロセッサを対象し、

前記パーサステップでは、前記固定小数点モードを切り替える旨の記述を前記ソースプログラムにおいて検出し、

前記コンパイラは、さらに、

前記パーサーステップにおいて前記固定小数点モードを切り替える旨の記述が検出された場合に、その記述に従って、固定小数点モードを切り替えるための機械語命令を挿入する固定小数点モード切替ステップを含むことを特徴とする請求項 1 記載のプログラム。

【請求項 3 2】 前記固定小数点モードを切り替える旨の記述は、対象となる関数に対応づけられ、

前記固定小数点モード切替ステップでは、対応する関数の先頭及び末尾に、それぞれ、固定小数点モードの退避及び復帰に相当する機械語命令を挿入することを特徴とする請求項 3 1 記載のプログラム。

【請求項 3 3】 前記最適化ステップは、前記ソースプログラムにおいて、特定箇所での実行時間が所定サイクル数だけ確保されるようなレイテンシを指定する旨の記述を検出し、検出した指定に従って、前記レイテンシが確保されるように機械語命令のスケジューリングを行うレイテンシ最適化サブステップを含むことを特徴とする請求項 1 記載のプログラム。

【請求項 3 4】 前記レイテンシ最適化サブステップでは、第 1 ラベルが付された第 1 機械語命令と第 2 レベルが付された第 2 機械語命令間を対象とした一定サイクル数のレイテンシを指定する旨の記述を検出すると、前記第 1 機械語命令が実行されてから前記第 2 機械語命令が実行されるまでに前記サイクル数だけ実行時間がかかるように、前記スケジューリングを行うことを特徴とする請求項 3 3 記載のプログラム。

【請求項 3 5】 前記レイテンシ最適化サブステップでは、所定のレジスタへのアクセスを対象とした一定サイクル数のレイテンシを指定する旨の記述を検出すると、前記レジスタにアクセスする機械語命令が実行されてから次に当該レジスタにアクセスする機械語命令が実行されるまでに前記サイクル数だけ実行時間がかかるように、前記スケジューリングを行う

ことを特徴とする請求項 3 3 記載のプログラム。

【請求項 3 6】 前記コンパイラは、さらに、前記演算処理定義情報で使用されている機械語命令を当該コンパイラが対象としている第 1 プロセッサとは異

なる第2プロセッサの機械語命令に置き換えるためのクラスライブラリを備えることを特徴とする請求項1記載のプログラム。

【請求項37】 コンパイラの対象となるソースプログラムにインクルードされるヘッダファイルが記録されたコンピュータ読み取り可能な記録媒体であって、

請求項2～36のいずれか1項に記載されたヘッダファイルが記録された記録媒体。

【請求項38】 コンパイラの対象となるソースプログラムにインクルードされるクラスライブラリが記録されたコンピュータ読み取り可能な記録媒体であって、

請求項36に記載されたクラスライブラリが記録された記録媒体。

【請求項39】 コンパイラの対象となるソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であって、

請求項2～36のいずれか1項に記載されたヘッダファイル及びクラスライブラリの少なくとも1つをインクルードしているソースプログラムが記録された記録媒体。

【請求項40】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であって、

対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を保持する手段と、

前記ソースプログラムを解析するパーサー手段と、

解析されたソースプログラムを中間コードに変換する中間コード変換手段と、

変換された中間コードを最適化する最適化手段と、

最適化された中間コードを機械語命令に変換するコード生成手段とを備え、

前記中間コード変換手段は、

前記中間コードのうち、前記演算処理定義情報で定義された演算処理を参照しているものがあるか否かを検出する検出部と、

あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換部とを含み、

前記最適化手段は、前記置換部で置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う

ことを特徴とするコンパイラ装置。

【請求項 4 1】 ソースプログラムを機械語プログラムに翻訳するコンパイラ方法であって、

前記プログラムは、

前記ソースプログラムを解析するパーサーステップと、

解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、

変換された中間コードを最適化する最適化ステップと、

最適化された中間コードを機械語命令に変換するコード生成ステップとを含み

前記中間コード変換ステップは、

前記中間コードのうち、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報において定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、

あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、

前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う

ことを特徴とするコンパイル方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、C++言語等の高級言語で記述されたソースプログラムを機械語プログラムに変換するコンパイラに関し、特に、コンパイラによる最適化に関する

【0002】

【従来の技術】

近年のプロセッサの高機能化に伴い、プロセッサが備える高い機能を有効に引き出すことが可能な性能の高いコンパイラが求められている。つまり、ターゲットプロセッサが備える高機能な専用命令を効率的に生成するコンパイラが求められている。

【 0 0 0 3 】

例えば、デジタル信号処理等のメディア処理に必要とされる各種固定小数点フォーマットでの演算命令を実行するプロセッサや、SIMD (Single Instruction Multiple Data) 型の命令を実行する高機能なプロセッサが開発されているが、そのようなプロセッサをターゲットとするコンパイラは、各種固定小数点フォーマットでの演算命令やSIMD型命令を効率的に生成することによってコードサイズ及び実行速度の点で最適化を図ることが求められる。

【 0 0 0 4 】

【発明が解決しようとする課題】

しかしながら、従来のコンパイラは、C++言語等の高級言語で記述されたソースプログラムに対しては、必ずしも、プロセッサが備える高機能な専用命令を効率的に生成しているとは言えない。そのために、コードサイズと実行速度の点で厳しい条件が求められるメディア処理等のアプリケーション開発においては、ユーザは、クリティカルな箇所については、アセンブラ命令で記述せざるを得ないのが現状である。ところが、アセンブラ命令でのプログラミングは、C++言語等の高級言語を用いた開発に比べ、開発工数が大きくなるだけでなく、保守性や移植性についても著しく劣るという問題がある。

【 0 0 0 5 】

また、従来のコンパイラでは、プロセッサが備える高機能な専用命令を生成する等の最適化処理は、コンパイラ自身に組み込まれている。つまり、ターゲットプロセッサの特徴を有効に活用して最適化を図る処理モジュールは、コンパイラ自身に組み込まれ、一体化されている。そのために、コンパイラの機能拡張を図る場合や、ターゲットプロセッサの仕様が変更された場合等においては、コンパイラ全体の再構築が必要とされ、その度に、コンパイラのバージョンアップ等を繰り返さなければならないという問題がある。

【0006】

そこで、本発明は、このような状況に鑑みてなされたものであり、プロセッサが備える高機能な専用命令を効率的に生成することができるコンパイラを提供することを目的とする。

また、コンパイラ自身のバージョンアップを頻繁に繰り返すことなく、機能拡張等の改良を行っていくことが可能なコンパイラ等を提供することをも目的とする。

【0007】

【課題を解決するための手段】

上記目的を達成するために、本発明に係るコンパイラ用プログラムは、ソースプログラムを機械語プログラムに翻訳するコンパイラ用のプログラムであって、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、前記プログラムは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記演算処理定義情報で定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

【0008】

例えば、本発明は、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、データとメソッドとからなるクラスが定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適

化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義されたクラスを参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うプログラムである。

【 0 0 0 9 】

ここで、前記クラスは、固定小数点型を定義するクラスであり、前記検出サブステップでは、前記固定小数点型のデータを使用している中間コードを検出してもよい。そして、前記クラスのメソッドは、前記固定小数点型のデータを対象とする演算子を定義するメソッドであり、前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換えてもよい。

【 0 0 1 0 】

また、本発明に係るコンパイラ用プログラムは、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、関数が定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義された関数を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

【 0 0 1 1 】

なお、本発明は、このようなコンパイラ用プログラムとして実現することができるだけでなく、このようなプログラムに含まれるステップを手段とするコンパイラ装置として実現したり、このような特徴的なコンパイラあるいはヘッダファイルが記録されたコンピュータ読み取り可能な記録媒体として実現することもできる。そして、そのようなプログラムやデータファイルは、CD-ROM等の記録媒体やインターネット等の伝送媒体を介して広く流通させることができるのは言うまでもない。

【 0 0 1 2 】

【発明の実施の形態】

以下、本発明に係るコンパイラの実施の形態について図面を用いて詳細に説明する。

本実施の形態におけるコンパイラは、C/C++言語等の高級言語で記述されたソースプログラムを特定のプロセッサ（ターゲット）が実行できる機械語プログラムに翻訳するクロスコンパイラであり、生成する機械語プログラムのコードサイズや実行時間に関する最適化の指示をきめ細かく指定することができるという特徴を有する。

【 0 0 1 3 】

[プロセッサ]

まず、本実施の形態におけるコンパイラの対象となるプロセッサの一例について、図1～図36を用いて説明する。

本実施の形態におけるコンパイラの対象となるプロセッサは、例えば、通常のマイコンに比べて実行可能な命令の並列性が高く、AVメディア系信号処理技術分野をターゲットとして開発された汎用プロセッサである。

【 0 0 1 4 】

図1は、そのようなプロセッサの概略ブロック図の一例である。このプロセッサ1は、32ビットを基本語長（ワード）とする演算装置であり、命令制御部10、デコード部20、レジスタファイル30、演算部40、I/F部50、命令メモリ部60、データメモリ部70、拡張レジスタ部80及びI/Oインターフ

エース部 9 0 から構成される。演算部 4 0 は、S I M D 型命令の演算を実行する算術論理・比較演算器 4 1 ~ 4 3、乗算・積和演算器 4 4、バレルシフタ 4 5、除算器 4 6 及び変換器 4 7 からなる。乗算・積和演算器 4 4 は、ビット精度を落とさないように、最長で 6 5 ビットで累算する。また、乗算・積和演算器 4 4 は、算術論理・比較演算器 4 1 ~ 4 3 と同様、S I M D 型命令の実行が可能である。更に、このプロセッサ 1 は、算術論理・比較演算命令が最大 3 並列実行可能である。

【 0 0 1 5 】

図 2 は、算術論理・比較演算器 4 1 ~ 4 3 の概略図を示す。算術論理・比較演算器 4 1 ~ 4 3 それぞれは、A L U 部 4 1 a、飽和处理部 4 1 b 及びフラグ部 4 1 c から構成される。A L U 部 4 1 a は、算術演算器、論理演算器、比較器、T S T 器からなる。対応する演算データのビット幅は、8 ビット(バイト。このときは、演算器を 4 並列で使用)、1 6 ビット(ハーフワード。このときは、演算器を 2 並列で使用)、3 2 ビットである(ワード。このときは、全演算器で 3 2 ビットデータ処理)。更に算術演算結果に対しては、フラグ部 4 1 c 等により、オーバーフローの検出とコンディションフラグの生成が行われる。各演算器、比較器、T S T 器の結果は、算術右シフト、飽和处理部 4 1 b による飽和、最大・最小値検出、絶対値生成処理が行われる。

【 0 0 1 6 】

図 3 は、バレルシフタ 4 5 の構成を示すブロック図である。バレルシフタ 4 5 は、セレクタ 4 5 a、4 5 b、上位バレルシフタ 4 5 c、下位バレルシフタ 4 5 d 及び飽和处理部 4 5 e から構成され、データの算術シフト(2 の補数体系のシフト)または、論理シフト(符号なしシフト)を実行する。通常は、3 2 ビットもしくは、6 4 ビットのデータを入出力としている。レジスタ 3 0 a、3 0 b に格納された被シフトデータに対して、別のレジスタまたは即値でシフト量が指定される。データは、左 6 3 ビット ~ 右 6 3 ビットの算術または論理シフトが行われ、入力ビット長で出力される。

【 0 0 1 7 】

また、バレルシフタ 4 5 は、S I M D 型命令に対して、8、1 6、3 2、6 4

ビットのデータをシフトすることができる。例えば、8ビットデータのシフトを4並列で処理することができる。

【 0 0 1 8 】

算術シフトは、2の補数体系のシフトであり、加算や減算時の小数点の位置合わせや、2のべき乗の乗算(2、2の2乗、2の(-1)乗、2の(-2)乗倍など)等のために行われる。

【 0 0 1 9 】

図4は、変換器47の構成を示すブロック図である。変換器47は、飽和ブロック(SAT)47a、BSEQブロック47b、MSKGENブロック47c、VSUMBブロック47d、BCNTブロック47e及びILブロック47fから構成される。

【 0 0 2 0 】

飽和ブロック(SAT)47aは、入力データに対する飽和处理を行う。32ビットデータを飽和处理するブロックを2つ持つことにより、2並列のSIMD型命令をサポートする。

【 0 0 2 1 】

BSEQブロック47bは、MSBから連続する0か1をカウントする。

MSKGENブロック47cは、指定されたビット区間を1、それ以外を0として出力する。

VSUMBブロック47dは、入力データを指定されたビット幅に区切り、その総和を出力する。

BCNTブロック47eは、入力データで1となっているビットの数をカウントする。

ILブロック47fは、入力データを指定されたビット幅に区切り、各データブロックを入れ換えた値を出力する。

【 0 0 2 2 】

図5は、除算器46の構成を示すブロック図である。除算器46は、被除数を64ビット、除数を32ビットとし、商と剰余を32ビットずつ出力する。商と剰余を求めるまでに34サイクルを必要とする。符号付き、符号なし、両方のデータを扱うことが可能である。ただし、被除数と除数において符号の有無の設定

は共通とする。その他、オーバーフローフラグ、0除算フラグを出力する機能を有する。

【0023】

図6は、乗算・積和演算器44の構成を示すブロック図である。乗算・積和演算器44は、2つの32ビット乗算器(MUL)44a、44b、3つの64ビット加算器(Adder)44c~44e、セレクタ44f及び飽和处理部(Saturation)44gから構成され、以下の乗算、積和演算を行う。

- ・ 32×32ビットのsignedの乗算、積和、積差演算
- ・ 32×32ビットのunsignedの乗算
- ・ 16×16ビットの2並列のsignedの乗算、積和、積差演算
- ・ 32×16ビットの2並列のsignedの乗算、積和、積差演算

これらの演算を整数、固定小数点フォーマット(h1、h2、w1、w2)のデータに対して行う。また、これらの演算に対し、丸め、飽和を行う。

【0024】

図7は、命令制御部10の構成を示すブロック図である。命令制御部10は、命令キャッシュ10a、アドレス管理部10b、命令バッファ10c~10e、ジャンプバッファ10f及びローテーション部(rotation)10gから構成され、通常時及び分岐時の命令供給を行う。128ビットの命令バッファを3つ(命令バッファ10c~10e)持つことにより、最大並列実行数に対応している。分岐処理に関しては、分岐実行前に、ジャンプバッファ10f等を介して、後述するTARレジスタに予め分岐先アドレスを格納する(settar命令)。TARレジスタに格納された分岐先アドレスを使用して、分岐を行う。

【0025】

なお、本実施の形態におけるコンパイラの対象となるプロセッサ1は、例えば、VLIWアーキテクチャを持つプロセッサである。ここで、VLIWアーキテクチャとは、1つの命令語中に複数の命令(ロード、ストア、演算、分岐など)を格納し、それらを全て同時に実行するアーキテクチャである。プログラマは、並列実行可能な命令を1つの発行グループとして記述することによって、その発行グループを並列処理させることができる。本明細書では、発行グループの区切り

を”;;”で示す。以下に表記例を示す。

(例 1)

```
mov r1, 0x23;;
```

この命令記述は、命令movのみを実行することを意味する。

(例 2)

```
mov r1, 0x38
```

```
add r0, r1, r2
```

```
sub r3, r1, r2;;
```

これらの命令記述は、命令mov、add、subを3並列で実行することを意味する。

命令制御部10は、発行グループを識別し、デコード部20に送る。デコード部20では、発行グループの命令を解析し、必要な資源を制御する。

【0026】

次に、このようなプロセッサ1が備えるレジスタについて説明する。

プロセッサ1のレジスタセットは、以下の表1に示される通りである。

【表1】

レジスタ名	ビット幅	本数	用途
R0~R31	32ビット	32本	汎用レジスタ。データメモリのポインタ、演算命令におけるデータの格納等に使 用します。
TAR	32ビット	1本	分岐用レジスタ。分岐時の分岐アドレスの格納に使 用します。
LR	32ビット	1本	リンク用レジスタ。
SVR	16ビット	2本	退避用レジスタ。コンディションフラグ(CFR)と各種モ ードを退避します。
M0~M1 (MH0:ML0~MH1:ML1)	64ビット	2本	演算用レジスタ。演算命令におけるデータの格納に 使用します。

【0027】

また、このようなプロセッサ1のフラグセット（後述する条件フラグレジスタ等で管理されるフラグ）は、以下の表2に示される通りである。

【表 2】

C0～C7	1ビット	8本	条件フラグ。条件成立・不成立を表します。
VC0～VC3	1ビット	4本	メディア処理拡張命令用条件フラグ。条件成立・不成立を表します。
OVS	1ビット	1本	オーバーフローフラグ。演算時のオーバーフローを検出します。
CAS	1ビット	1本	キャリーフラグ。演算時のキャリーを検出します。
BPO	5ビット	1本	ビット位置指定。マスク処理命令時に処理対象となるビット位置を指定します。
ALN	2ビット	1本	バイトアライン指定。
FXP	1ビット	1本	固定小数点演算モード。
UDR	32ビット	1本	未定義レジスタ

【0028】

図8は、汎用レジスタ（R0～R31）30aの構造を示す図である。汎用レジスタ（R0～R31）30aは、実行対象となっているタスクのコンテキストの一部を構成し、データまたはアドレスを格納する32ビットのレジスタ群である。なお、汎用レジスタR30およびR31は、それぞれグローバルポインタ、スタックポインタとして、ハードウェアが使用する。

【0029】

図9は、リンクレジスタ（LR）30cの構造を示す図である。なお、このリンクレジスタ（LR）30cと関連して、このプロセッサ1は、図示されていない退避レジスタ（SVR）も備える。リンクレジスタ（LR）30cは、関数コール時のリターンアドレスを格納する32ビットのレジスタである。なお、退避レジスタ（SVR）は、関数コール時の条件フラグレジスタのコンディションフラグ（CFR.CF）を退避する16ビットのレジスタである。リンクレジスタ（LR）30cは、後述する分岐レジスタ（TAR）と同様に、ループ高速化にも使用される。下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

【0030】

例えば、call(brl, jmp1)命令を実行した場合には、このプロセッサ1は、リンクレジスタ（LR）30cに戻りアドレスを退避し、退避レジスタ（SVR）にコンディションフラグ（CFR.CF）を退避する。また、jmp命令を実行した場合には、リンクレジスタ（LR）30cから戻りアドレス（分岐先アドレス）を取り出し

、プログラムカウンタ（PC）を復帰させる。さらに、ret(jmpr)命令を実行した場合には、リンクレジスタ（LR）30cから分岐先アドレス(戻りアドレス)を取り出し、プログラムカウンタ（PC）に格納(復帰)する。さらに、退避レジスタ（SVR）からコンディションフラグを取り出し、条件フラグレジスタ（CFR）32のコンディションフラグ領域CFR.CFに格納(復帰)する。

【0031】

図10は、分岐レジスタ（TAR）30dの構造を示す図である。分岐レジスタ（TAR）30dは、分岐ターゲットアドレスを格納する32ビットのレジスタである。主に、ループの高速化に用いられる。下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

【0032】

例えば、jmp,jloop命令を実行した場合には、プロセッサ1は、分岐レジスタ（TAR）30dから分岐先アドレスを取り出し、プログラムカウンタ（PC）に格納する。分岐レジスタ（TAR）30dに格納されたアドレスの命令が分岐用命令バッファに格納されている場合は、分岐ペナルティが0になる。分岐レジスタ（TAR）30dにループの先頭アドレスを格納しておくことでループを高速化することができる。

【0033】

図11は、プログラム状態レジスタ（PSR）31の構造を示す図である。プログラム状態レジスタ（PSR）31は、実行対象となっているタスクのコンテキストの一部を構成し、以下に示されるプロセッサ状態情報を格納する32ビットのレジスタである。

【0034】

ビットSWE：VMP（Virtual Multi-Processor）のLP（Logical Processor）切替えイネーブルを示す。「0」はLP切替え不許可を示し、「1」はLP切替え許可を示す。

【0035】

ビットFXP：固定小数点モードを示す。「0」はモード0（MSBとMSBから1ビット目との間に小数点があるものとして演算をするモード。以下、「_

1系」ともいう。)を示し、「1」はモード1 (MSBから1ビット目とMSBから2ビット目との間に小数点があるものとして演算をするモード。—2系」ともいう。)を示す。

【0036】

ビットIH: 割込み処理フラグであり、マスカブル割込み処理中であることを示す。「1」は割込み処理中であることを示し、「0」は割込み処理中でないことを示す。割込みが発生すると自動的にセットされる。rti命令で割込みから復帰したところが、他の割込み処理中かプログラム処理中であるのかを見分けるために使用される。

【0037】

ビットEH: エラーまたはNMIを処理中であることを示すフラグである。「0」はエラー/NMI割込み処理中でないことを示し、「1」はエラー/NMI割込み処理中であることを示す。EH=1のとき、非同期エラーまたはNMIが発生した場合は、マスクされる。また、VMPイネーブル時はVMPのプレート切り替えがマスクされる。

【0038】

ビットPL[1:0]: 特権レベルを示す。「00」は特権レベル0、つまり、プロセッサアブストラクションレベルを示し、「01」は特権レベル1 (設定できない)を示し、「10」は特権レベル2、つまり、システムプログラムレベルを示し、「11」は特権レベル3、つまり、ユーザプログラムレベルを示す。

【0039】

ビットLPIE3: LP固有割込み3イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE2: LP固有割込み2イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE1: LP固有割込み1イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE0: LP固有割込み0イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビット A E E : ミスアライメント例外イネーブルを示す。「1」はミスアライメント例外許可を示し、「0」はミスアライメント例外不許可を示す。

ビット I E : レベル割込みイネーブルを示す。「1」はレベル割込み許可を示し、「0」はレベル割込み不許可を示す。

ビット I M [7 : 0] : 割込みマスクを示す。レベル 0 ~ 7 まで定義され、個々のレベルでマスクすることができる。レベル 0 が最も高いレベルとなる。I M によりマスクされていない割込み要求のうち最も高いレベルを持った割込み要求のみがプロセッサ 1 に受理される。割込み要求を受理すると受理したレベル以下のレベルはハードウェアで自動的にマスクされる。I M [0] はレベル 0 のマスクであり、I M [1] はレベル 1 のマスクであり、I M [2] はレベル 2 のマスクであり、I M [3] はレベル 3 のマスクであり、I M [4] はレベル 4 のマスクであり、I M [5] はレベル 5 のマスクであり、I M [6] はレベル 6 のマスクであり、I M [7] はレベル 7 のマスクである。

r e s e r v e d : 予約ビットを示す。常に 0 が読み出される。書き込む時は 0 を書き込む必要がある。

【 0 0 4 0 】

図 1 2 は、条件フラグレジスタ (C F R) 3 2 の構造を示す図である。条件フラグレジスタ (C F R) 3 2 は、実行対象となっているタスクのコンテキストの一部を構成する 3 2 ビットのレジスタであり、コンディションフラグ(条件フラグ)、オペレーションフラグ(演算フラグ)、ベクタコンディションフラグ(ベクタ条件フラグ)、演算命令用ビット位置指定フィールド、SIMDデータアライン情報フィールドから構成される。

【 0 0 4 1 】

ビット A L N [1 : 0] : アラインモードを示す。v a l n v c 命令のアラインモードを設定する。

ビット B P O [4 : 0] : ビットポジションを示す。ビット位置指定の必要な命令で使用する。

ビット V C 0 ~ V C 3 : ベクタ条件フラグである。L S B 側のバイトあるいはハーフワードから順に V C 0 に対応し、M S B 側が V C 3 に対応する。

ビット O V S : オーバーフローフラグ(サマリー)である。飽和発生やオーバー

フロー検出でセットされる。検出されなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

ビット C A S : キャリーフラグ(サマリー)である。addc命令でキャリーまたはsubc命令でボローが発生した場合セットされる。addc命令でキャリーもしくはsubc命令でボローが発生しなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

ビット C 0 ~ C 7 : コンディションフラグである。条件付き実行命令における条件 (TRUE / FALSE) を示す。条件付き命令の条件とビット C 0 ~ C 7 との対応は、命令に含まれるプレディケート・ビットによって決定される。なお、フラグ C 7 は常に値が 1 である。フラグ C 7 への FALSE 条件の反映 (0 書き込み) は無視される。

reserved : 予約ビットを示す。常に 0 が読み出される。書き込む時は 0 を書き込む必要がある。

【 0 0 4 2 】

図 1 3 は、アキュムレータ (M 0 , M 1) 3 0 b の構造を示す図である。このアキュムレータ (M 0 , M 1) 3 0 b は、実行対象となっているタスクのコンテキストの一部を構成し、図 1 3 (a) に示される 3 2 ビットレジスタ MH 0 - MH 1 (乗除算・積和用レジスタ(上位 3 2 ビット)) と、図 1 3 (b) に示される 3 2 ビットレジスタ ML 0 - ML 1 乗除算・積和用レジスタ(下位 3 2 ビット)とからなる。

【 0 0 4 3 】

レジスタ MH 0 - MH は、乗算命令では結果の上位 3 2 ビットを格納するのに使用される。積和命令ではアキュムレータの上位 3 2 ビットとして使用される。また、ビットストリームを取り扱う場合に汎用レジスタと組み合わせて使用することができる。レジスタ ML 0 - ML 1 は、乗算命令では結果の下位 3 2 ビットを格納するのに使用される。積和命令ではアキュムレータの下位 3 2 ビットとして使用される。

【 0 0 4 4 】

図 1 4 は、プログラムカウンタ (P C) 3 3 の構造を示す図である。このプログラムカウンタ (P C) 3 3 は、実行対象となっているタスクのコンテキストの

一部を構成し、実行中の命令のアドレスを保持する 3 2 ビットのカウンタである。下位 1 ビットは常に 0 が格納される。

【 0 0 4 5 】

図 1 5 は、P C 退避用レジスタ (I P C) 3 4 の構造を示す図である。この P C 退避用レジスタ (I P C) 3 4 は、実行対象となっているタスクのコンテキストの一部を構成する 3 2 ビットのレジスタであり、下位 1 ビットは常に 0 が読み出されるが、書き込み時には 0 を書き込む必要がある。

【 0 0 4 6 】

図 1 6 は、P S R 退避用レジスタ (I P S R) 3 5 の構造を示す図である。この P S R 退避用レジスタ (I P S R) 3 5 は、実行対象となっているタスクのコンテキストの一部を構成し、プログラム状態レジスタ (P S R) 3 1 を退避するための 3 2 ビットのレジスタであり、プログラム状態レジスタ (P S R) 3 1 の予約ビットに対応する部分は常に 0 が読み出されるが、書き込み時には 0 を書き込む必要がある。

【 0 0 4 7 】

次に、本実施の形態におけるコンパイラの対象となるプロセッサ 1 のメモリ空間について説明する。例えば、プロセッサ 1 では、4 G B のリニアなメモリ空間を 3 2 分割し、1 2 8 M B 単位の空間に命令 S R A M (Static RAM) とデータ S R A M が割り当てられる。この 1 2 8 M B の空間を 1 ブロックとして、S A R (S R A M Area Register) にアクセスしたいブロックを設定する。アクセスされたアドレスが S A R で設定された空間である場合は、直接命令 S R A M / データ S R A M に対してアクセスを行うが、S A R で設定された空間でない場合は、バスコントローラ (B C U) に対してアクセス要求を出す。B C U にはオン・チップ・メモリ (O C M) 、外部メモリ、外部デバイス、I / O ポート等が接続されており、それらのデバイスに対して読み書きを行うことができる。

【 0 0 4 8 】

図 1 7 は、本実施の形態におけるコンパイラの対象となるプロセッサ 1 のパイプライン動作を示すタイミング図である。このプロセッサ 1 は、本図に示されるように、例えば、基本的に命令フェッチ、命令割り当て(ディスパッチ)、デコー

ド、実行、書き込みの 5 段パイプラインで構成されている。

【 0 0 4 9 】

図 1 8 は、このようなプロセッサ 1 による命令実行時の各パイプライン動作を示すタイミング図である。命令フェッチステージでは、プログラムカウンタ（PC）3 3 で指定されるアドレスの命令メモリをアクセスし、命令を命令バッファ 1 0 c ～ 1 0 e 等に転送する。命令割り当てステージでは、分岐系命令に対する分岐先アドレス情報の出力、入力レジスタ制御信号の出力、可変長命令の割り当てを行い、命令をインストラクションレジスタ（IR）に転送する。デコードステージでは、IR をデコード部 2 0 に入力し、演算器制御信号、メモリアクセス信号を出力する。実行ステージでは、演算を実行、演算結果をデータメモリか汎用レジスタ（R 0 ～ R 3 1）3 0 a に出力する。書き込みステージでは、データ転送、演算結果を汎用レジスタに格納する。

【 0 0 5 0 】

本実施の形態におけるコンパイラの対象となるプロセッサ 1 は、例えば、VLIWアーキテクチャにより上記の処理を最高 3 並列で行うことができる。したがって、図 1 8 に示された動作については、本プロセッサ 1 は、図 1 9 に示されるタイミングで並列に実行する。

【 0 0 5 1 】

次に、以上のように構成されたプロセッサ 1 の命令セットの例について説明する。

以下の表 3 ～ 表 5 は、本実施の形態におけるコンパイラの対象となるプロセッサ 1 が実行する命令をカテゴリー別に分類した表である。

【表 3】

カテゴリー	演算器	命令オペコード
メモリ転送命令(ロード)	M	ld, ldh, ldhu, ldb, ldbu, ldp, ldhp, ldbp, ldbh, ld buh, ldbhp, ldbuhp
メモリ転送命令(ストア)	M	st, sth, stb, stp, sthp, stbp, stbh, stbhp
メモリ転送命令(その他)	M	dpref, ldstb
外部レジスタ転送命令	M	rd, rde, wt, wte
分岐命令	B	br, brl, call, jmp, jmp1, jmpr, ret, jmpf, jloop, s etbb, setlr, settar
ソフトウェア割込み命令	B	rti, pi0, pi0l, pi1, pi1l, pi2, pi2l, pi3, pi3l, pi 4, pi4l, pi5, pi5l, pi6, pi6l, pi7, pi7l, sc0, sc1, sc2, sc3, sc4, sc5, sc6, sc7
VMP/割込み制御命令	B	intd, inte, vmpsleap, vmpsus, vmpswd, vmpswe, vm pwait
算術演算命令	A	abs, absvh, absvw, add, addarvw, addc, addmsk, ad ds, addsr, addu, addvh, addvw, neg, negvh, negvw, rsub, sladd, s2add, sub, subc, submsk, subs, subv h, subvw, max, min
論理演算命令	A	and, andn, or, sethi, xor, not
比較命令	A	cmpCC, cmpCCa, cmpCCn, cmpCCo, tstn, tstna, tstn n, tstno, tstz, tstza, tstzn, tstzo
転送命令	A	mov, movcf, mvclcas, mvclcvs, setlo, vcchk
NOP命令	A	nop
シフト命令1	S1	asl, aslvh, aslvw, asr, asrvh, asrvw, lsl, lsr, ro l, ror
シフト命令2	S2	aslp, aslpvw, asrp, asrpvw, lslp, lsrp

【表 4】

カテゴリー	演算器	命令オペコード
抽出命令	S2	ext, extb, extbu, exth, exthu, extr, extru, extu
マスク命令	C	msk, mskgen
飽和命令	C	sat12, sat9, satb, satbu, sath, satw
変換命令	C	valn, valn1, valn2, valn3, valnvc1, valnvc2, valnvc3, valnvc4, vhpkb, vhpkh, vhunpkb, vhunpkh, vintlhb, vintlhh, vintlhb, vintlhh, vlpkb, vlpkb u, vlpkh, vlpkhu, vlunpkb, vlunpkbu, vlunpkh, vlunpkhu, vstovb, vstovh, vunpk1, vunpk2, vxchngh, vexth
ビットカウント命令	C	bcnt1, bseq, bseq0, bseq1
その他	C	byterev, extw, mskbrvb, mskbrvh, rndvh, movp
乗算命令1	X1	fmulhh, fmulhhr, fmulhw, fmulhww, hmul, lmul
乗算命令2	X2	fmulww, mul, mulu
積和命令1	X1	fmachh, fmachhr, fmachw, fmachww, hmac, lmac
積和命令2	X2	fmacww, mac
積差命令1	X1	fmsuhh, fmsuhhr, fmsuhw, fmsuww, hmsu, lmsu
積差命令2	X2	fmsuww, msu
割算命令	DIV	div, divu
デバッグ命令	DBGM	dbgm0, dbgm1, dbgm2, dbgm3

【表 5】

カテゴリー	演算器	命令オペコード
SIMD 算術演算命令	A	vabshvh, vaddb, vaddh, vaddhvc, vaddhvh, vaddrh vc, vaddsb, vaddsh, vaddsrh, vaddsrh, vasubb, vc chk, vhaddh, vhaddhvh, vhsubb, vhsubhvh, vladdh , vladdhvh, vlsubh, vlsubhvh, vnegb, vnegh, vneg hvh, vsaddb, vsaddh, vsgrh, vsrsubb, vsrsubh, vs subb, vssubb, vsubb, vsubh, vsubhvh, vsubsh, vsu mh, vsumh2, vsumrh2, vxaddh, vxaddhvh, vxsubh, v xsubhvh, vmaxb, vmaxh, vminb, vminh, vmovt, vsel
SIMD 比較命令	A	vcmpqeb, vcmpqeh, vcmpgeb, vcmpgeh, vcmpgtb, vc mpgth, vcmpleb, vcmpleh, vcmpltb, vcmplth, vcmp neb, vcmpneh, vscmpqeb, vscmpqeh, vscmpgeb, vscmpgeh, vscmpg tb, vscmpgth, vscmpleb, vscmpleh, vscmpltb, vsc mplth, vscmpneb, vscmpneh
SIMD シフト命令1	S1	vaslb, vaslh, vaslvh, vasrb, vasrh, vasrvh, vlsl b, vlslh, vlslrb, vlslrh, vroib, vroih, vroib, vro rh
SIMD シフト命令2	S2	vasl, vaslvw, vasr, vasrvw, vlsl, vlslr
SIMD 飽和命令	C	vsath, vsath12, vsath8, vsath8u, vsath9
SIMD その他の命令	C	vabssumb, vrndvh
SIMD 乗算命令	X2	vfmulh, vfmulhr, vfmulw, vhfmulh, vhfmulhr, vhf mulw, vhmul, vlfmulh, vlfmulhr, vlfmulw, vlmul, vmul, vpfmulhww, vxfmulh, vxfmulhr, vxfmulw, vx mul
SIMD 積和命令	X2	vfmach, vfmachr, vfmacw, vhfmach, vhfmacwr, vhf macw, vhmach, vlfmach, vlfmacwr, vlfmacw, vlmach, vmach, vpfmachww, vxfmach, vxfmachr, vxfmachw, vx mach
SIMD 積差命令	X2	vfmsuh, vfmsuw, vhfmsuh, vhfmsuw, vhmsh, vlfmsu h, vlfmsuw, vlmsu, vmsu, vxfmsuh, vxfmsuw, vxmsu

【0052】

なお、表中の「演算器」は、その命令が使用する演算器を示す。演算器の略号の意味は次の通りである。つまり、「A」はALU命令、「B」は分岐命令、「C」は変換命令、「DIV」は除算命令、「DBGM」はデバッグ命令、「M」はメモリアクセス命令、「S1」、「S2」はシフト命令、「X1」、「X2」は乗算命令を意味する。

【0053】

図20は、このようなプロセッサ1が実行する命令のフォーマットの例を示す図である。そのフォーマットには、図20(a)に示される16ビット命令フォ

フォーマットと、図20(b)に示される32ビット命令フォーマットとがある。

【0054】

なお、図中における略号の意味は次の通りである。つまり、「E」はエンドビット（並列実行の境界）、「F」はフォーマットビット（00、01、10：16ビット命令フォーマット、11：32ビット命令フォーマット）、「P」はプレディケート（実行条件：8個の条件フラグC0～C7のいずれかを指定）、「OP」はオペコードフィールド、「R」はレジスタフィールド、「I」は即値フィールド、「D」ディスプレイメントフィールドを意味する。なお、「E」フィールドはVLIWに特有のもので、E=0の命令は次の命令と並列に実行される。つまり、「E」フィールドによって並列度が可変のVLIWを実現している。

【0055】

図21～図36は、プロセッサ1が実行する命令の概略的な機能を説明する図である。つまり、図21は、カテゴリー「ALUadd（加算）系」に属する命令を説明する図であり、図22は、カテゴリー「ALUsub（減算）系」に属する命令を説明する図であり、図23は、カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図であり、図24は、カテゴリー「CMP（比較演算）系」に属する命令を説明する図であり、図25は、カテゴリー「mul（乗算）系」に属する命令を説明する図であり、図26は、カテゴリー「mac（積和演算）系」に属する命令を説明する図であり、図27は、カテゴリー「msu（積差演算）系」に属する命令を説明する図であり、図28は、カテゴリー「MEMld（メモリ読み出し）系」に属する命令を説明する図であり、図29は、カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図であり、図30は、カテゴリー「BRA（分岐）系」に属する命令を説明する図であり、図31は、カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図であり、図32は、カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図であり、図33は、カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図であり、図34は、カテゴリー「CNV（一般変換）系」に属する命令を説明する図であり、図35は、カテゴリー「SATvlpk（飽和处理）系」に属する命令を説明する図であり、図36は、カテゴリー「ETC（その他）系」に属する命

令を説明する図である。

【 0 0 5 6 】

これらの図において、項目「S I M D」は、その命令の型（S I S D（SINGLE）かS I M Dかの区別）を示し、項目「サイズ」は、演算の対象となる個々のオペランドのサイズを示し、項目「命令」は、その命令のオペコードを示し、項目「オペランド」は、その命令のオペランドを示し、項目「C F R」は、条件フラグレジスタの変化を示し、項目「P S R」は、プロセッサ状態レジスタの変化を示し、項目「代表的な動作」は、動作の概要を示し、項目「演算器」は、使用される演算器を示し、項目「3 1 1 6」は、命令のサイズを示す。

【 0 0 5 7 】

以下に、後述する具体例で使用される主な命令に対するプロセッサ 1 の動作を説明する。

andn Rc,Ra,Rb

RaとRbの反転の論理積をとり、Rcに格納する。

asl Rb,Ra,I5

Raを即値(I5)ビット数だけ算術左シフトする。

and Rb,Ra,I8

Raと値(I8)の論理積をとり、Rbに格納する。

bseq0 Rb,Ra

RaのMSBから連続する 0 の数を数えて、Rbに格納する。

【 0 0 5 8 】

bseq1 Rb,Ra

RaのMSBから連続する 1 の数を数えて、Rbに格納する。

bseq Rb,Ra

RaのMSBの 1 bit下から連続する符号ビットの数を数えて、Rbに格納する。Raが 0 の場合は、0を出力する。

bcnt1 Rb,Ra

Raの 1 の数を数えて、Rbに格納する。

extr Rc,Ra,Rb

Rbでbit位置を指定し、Raの内容の一部を取り出し、符号拡張してRcに格納する。

【 0 0 5 9 】

extru Rc,Ra,Rb

Rbでbit位置を指定し、Raの内容の一部を取り出し、符号拡張せずにRcに格納する。

fmulhh Mm,Rc,Ra,Rb

Ra,Rb,Rcを 16bit 値として扱い、Mm（乗算用アキュムレータ）を32bit値として扱う。RaとRbを固定小数点乗算する。結果をMmとRcに格納する。結果が符号付き 32 bitで表せない場合飽和する。

fmulhw Mm,Rc,Ra,Rb

RaとRbを 16bit 値として扱い、Mm,Rcを32bit値として扱う。RaとRbを固定小数点乗算する。結果をMmとRcに格納する。結果が符号付き 32 bitで表せない場合飽和する。

mul Mm,Rc,Ra,Rb

RaとRbを整数乗算する。結果をMmとRcに格納する。

【 0 0 6 0 】

mac Mm,Rc,Ra,Rb,Mn

RaとRbを整数乗算し、Mnと加算する。結果をMmとRcに格納する。

mov Rb,Ra

RaをRbに転送する。

or Rc,Ra,Rb

RaとRbの論理和をとり、Rcに格納する。

rde C0:C1,Rb,(Ra)

Raを外部レジスタ番号として、その外部レジスタの値をRbに読み込む。読み込みの成功・失敗をC0とC1（それぞれ、条件フラグ）に出力し、失敗時は拡張レジスタエラー例外が発生する。

wte C0:C1,(Ra),Rb

Raを外部レジスタ番号として、Rbの値をその外部レジスタに書き出する。書き

出しの成功・失敗をC0とC1に出力し、失敗時は拡張レジスタエラー例外が発生する。

vaddh Rc,Ra,Rb

各レジスタをハーフワードベクタ形式で扱う。RaとRbを加算(SIMD straight)する。

【 0 0 6 1 】

[コンパイラ]

次に、以上のプロセッサ1をターゲットとする本実施の形態におけるコンパイラについて説明する。

図37は、本実施の形態におけるコンパイラ100の構成を示す機能ブロック図である。このコンパイラ100は、C/C++言語等の高級言語で記述されたソースプログラム101を、上述のプロセッサ1をターゲットプロセッサとする機械語プログラム105に変換するクロスコンパイラであり、パーソナルコンピュータ等のコンピュータ上で実行されるプログラムによって実現され、大きく分けて、パーサー部110と、中間コード変換部120と、最適化部130と、コード生成部140とから構成される。

【 0 0 6 2 】

なお、本コンパイラ100には、上記プロセッサ1に特有の専用命令を効率的に生成するためのヘッダファイル（演算子定義ファイル102及び組み込み関数定義ファイル103）が準備されている。ユーザは、ソースプログラム101において、これらのヘッダファイルをインクルードすることで、プロセッサ1に特化した（最適化された）機械語プログラム105を得ることができる。

【 0 0 6 3 】

演算子定義ファイル102は、図38～図68のリスト例に示されるように、固定小数点及びSIMD型のデータを対象とする演算子を定義したクラスを定義しているヘッダファイルである。図38～図40は、ヘッダファイルのうち、主に、モード0（__1系）の16ビット固定小数点のデータを対象とする演算子（オペレータ）を定義している箇所のリストであり、図41～図42は、ヘッダファイルのうち、主に、モード0（__1系）の32ビット固定小数点のデータを対

象とする演算子（オペレータ）を定義している箇所のリストであり、図 4 3～図 4 5 は、ヘッダファイルのうち、主に、モード 1（__2 系）の 1 6 ビット固定小数点のデータを対象とする演算子（オペレータ）を定義している箇所のリストであり、図 4 5～図 4 7 は、ヘッダファイルのうち、主に、モード 1（__2 系）の 3 2 ビット固定小数点のデータを対象とする演算子（オペレータ）を定義している箇所のリストであり、図 4 8～図 6 8 は、その他の関数を定義している箇所のリストである。

【 0 0 6 4 】

組み込み関数定義ファイル 1 0 3 は、図 6 9～図 7 2 のリスト例に示されるように、プロセッサ 1 に特有の機械語命令に置き換えるための各種演算を行う関数を定義しているヘッダファイルである。図 6 9～図 7 1 は、ヘッダファイルのうち、1 つの機械語命令に置き換える関数を定義している箇所のリストであり、図 7 2 は、ヘッダファイルのうち、2 以上の機械語命令（機械語命令列）に置き換える関数を定義している箇所のリストである。

【 0 0 6 5 】

なお、これらの定義ファイル 1 0 2 及び 1 0 3 において、asm(…) {…} (…) は、最適化 asm と呼ばれる組み込みアセンブラ命令であり、以下の通り、処理される。つまり、最適化 asm 文の記述形式は、

【 0 0 6 6 】

```
asm(<<ロード式並び>>){
    <<最適化制御情報>>
    <<命令指定部>>
}( <<ストア式並び>>);
```

【 0 0 6 7 】

である。ここで、「ロード式並び」は、ロード式を記述する箇所であり、「ロード式」は、レジスタに C 言語での変数や四則演算などの式の結果を格納するための式であり、「レジスタ指定識別子 = 代入式」のように記述され、右辺に示される値を左辺に示される識別子に転送することを意味し、「ストア式並び」は、ストア式を記述する箇所であり、「ストア式」は、「単項式 = レジスタ指定

識別子」のように記述され、単項式で表される左辺値にレジスタ指定識別子で表されるレジスタの値を代入することを意味する。

【 0 0 6 8 】

パーサー部 1 1 0 は、コンパイルの対象となるソースプログラム 1 0 1（インクルードされるヘッダファイルを含む）に対して、予約語（キーワード）等を抽出して字句解析する前置処理部であり、通常のコンパイラが備える解析機能に加えて、固定小数点についてのモードの切り替えをサポートする固定小数点モード切替部 1 1 1 を有する。固定小数点モード切替部 1 1 1 は、ソースプログラム 1 0 1 中に、固定小数点モードを退避・復帰させるプリAGMA指令（例えば、`#pragma _save_fxpmode func`等）を検出すると、プロセッサ 1 の P S R 3 1 のビット F X P の退避・復帰を行う機械語命令を生成する。これによって、ユーザは、固定小数点のモード 0 及びモード 1 での演算が混在したプログラミングが可能となる。

【 0 0 6 9 】

なお、「プリAGMA（又は、プリAGMA指令）」とは、ソースプログラム 1 0 1 中にユーザが任意に指定（配置）することができるコンパイラ 1 0 0 への指示であり、「`#pragma`」で始まる文字列である。

【 0 0 7 0 】

中間コード変換部 1 2 0 は、パーサー部 1 1 0 から渡されたソースプログラム 1 0 1 の各ステートメントを中間コードに変換する処理部であり、中間コード生成部 1 2 1 と機械語命令置換部 1 2 2 とから構成される。中間コード生成部 1 2 1 は、ソースプログラム 1 0 1 の各ステートメントを一定規則に基づいて中間コードに変換する。ここで、中間コードは、典型的には、関数呼び出しの形式で表現されるコード（例えば、「`+(int a, int b)`」を示すコード；「整数 a に整数 b を加算する」ことを示す。）である。ただし、中間コードには、このような関数呼び出し形式のコードだけでなく、プロセッサ 1 の機械語命令も含まれる。

【 0 0 7 1 】

機械語命令置換部 1 2 2 は、中間コード生成部 1 2 1 で生成された中間コードのうち、関数呼び出し形式の中間コードに対して、演算子定義ファイル 1 0 2 及

び組み込み関数定義ファイル 1 0 3 を参照することで、それらの定義ファイルで定義された演算子（演算対象のデータの型も含む）又は組み込み関数に一致した中間コードについては、内部に保持する置換テーブル 1 2 2 a 又はこれらの定義ファイルで定義されたアセンブラ命令に従って、対応する機械語命令（又は機械語命令列）に置き換え、最適化部 1 3 0 に出力する。これによって、これらの中間コードについては、関数呼び出しの形式ではなく、機械語命令の形式で最適化部 1 3 0 に渡されるので、最適化部 1 3 0 での各種最適化が可能となる。

【 0 0 7 2 】

なお、置換テーブル 1 2 2 a は、予め予約された演算子による演算及び関数に対応する機械語命令（又は機械語命令列）を格納したテーブルである。また、機械語命令置換部 1 2 2 は、中間コード生成部 1 2 1 から渡された中間コードのうち、機械語命令については、そのまま最適化部 1 3 0 に出力する。

【 0 0 7 3 】

最適化部 1 3 0 は、中間コード変換部 1 2 0 から出力された中間コードのうち、機械語命令について、命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の処理を行うことで、（１）実行速度の向上を優先した最適化、（２）コードサイズの削減を優先した最適化、（３）実行速度とコードサイズの両方の最適化、の中からユーザによって選択された種類の最適化を実行する処理部であり、一般的な最適化（「ループアンローリング」、「i f 変換」、「ペアメモリアクセス命令の生成」など）に加えて、本コンパイラ 1 0 0 に特有の最適化を行う処理部（引数最適化部 1 3 1、型変換最適化部 1 3 2 及びレイテンシ最適化部 1 3 3）を有する。

【 0 0 7 4 】

引数最適化部 1 3 1 は、組み込み関数（extr、extru など）の引数に応じて、適切な命令又はシーケンス（アルゴリズム）を生成する処理部である。例えば、引数がすべて定数の場合には、それら定数を畳み込んで得られる定数値をオペランドとする機械語命令を生成し、引数の一部が定数の場合には、即値オペランドの機械語命令を生成し、引数のすべてが変数の場合には、レジスタオペランドの命令列を生成する。

【 0 0 7 5 】

型変換最適化部 1 3 2 は、ソースプログラム 1 0 1 中での一定表記に基づいて、異なる型間の演算を効率化する処理部である。例えば、1 6 ビットのデータと 1 6 ビットのデータとの乗算結果を 3 2 ビットのデータとして保持しておきたい場合に、ソースプログラム 1 0 1 において一定表記がなされているときには、そのような型変換を伴った乗算を行う 1 つの機械語命令（「fmulhw」など）を生成する。

【 0 0 7 6 】

レイテンシ最適化部 1 3 3 は、ソースプログラム 1 0 1 に組み込まれたアセンブラ命令中でのレイテンシに関する指示（サイクル数の指定）に基づいて、特定の区間あるいは特定の動作が指定されたサイクル数だけ実行時間がかかるように、機械語命令を並べる。これによって、従来であればプログラマが必要個数の n o p 命令を挿入していた作業が不要になるとともに、n o p 命令以外の他の機械語命令を挿入することによる最適化が可能となる。

【 0 0 7 7 】

なお、「ループアンローリング」とは、ループのイタレーション(繰り返し)を展開し、複数のイタレーションを同時に実行させるために、ペアメモリアクセス命令(ldp/stp/ldhp/sthp等)を生成し、これによって、ループの並列実行可能性を向上させる最適化である。また、「if変換」とは、条件付き実行機構の命令（命令に含まれる条件（プレディケート）がプロセッサ 1 の状態（コンディションフラグ）と一致している場合にだけ実行される命令）を生成することにより分岐構造を除去する最適化である。また、「ペアメモリアクセス命令の生成」とは、ペアレジスタ(2本の連続するレジスタ)を対象にメモリアクセスする命令(ldp/stp/ldhp/sthp等)を生成する最適化である。

【 0 0 7 8 】

また、最適化部 1 3 0 は、関数呼び出し形式の中間コードのうち、機械語命令に展開されないものについては、上述のような機械語命令レベルでの最適化処理を施すことができないので、そのままコード生成部 1 4 0 に出力する。

【 0 0 7 9 】

コード生成部 1 4 0 は、最適化部 1 3 0 から出力された中間コード（関数呼び出し形式のコード及び最適化された機械語命令を含む）に対して、内部に保持する変換テーブル等を参照することで、全てのコードを機械語命令に置き換えることで、機械語プログラム 1 0 5 を生成する。

【 0 0 8 0 】

次に、以上のように構成されたコンパイラ 1 0 0 の特徴的な動作について、具体的な例を示しながら説明する。

図 7 3 は、機械語命令置換部 1 2 2 の動作を示すフローチャートである。機械語命令置換部 1 2 2 は、中間コード生成部 1 2 1 で生成された中間コードのうち、関数呼び出し形式のコードについて、演算子定義ファイル 1 0 2 で定義された演算子（演算対象のデータ型を含む）及び組み込み関数定義ファイル 1 0 3 で定義された関数と一致するか否かを判断し（ステップ S 1 0 1）、一致する場合には（ステップ S 1 0 1 で Y e s）、内部に保持する置換テーブル 1 2 2 a 又はこれらの定義ファイル 1 0 2 及び 1 0 3 で定義されたアセンブラ命令にしたがって、機械語命令に置き換える（ステップ S 1 0 2）という処理を繰り返す（ステップ S 1 0 0 ～ S 1 0 3）。

【 0 0 8 1 】

具体的には、演算子定義ファイル 1 0 2 での定義により、異なる型間の暗黙の型変換のルール等が規定され（コンストラクタの定義による）、4 種類の固定小数点の型、つまり、

- ・ 「FIX16_1」；第 1 4 ビットと第 1 5 ビット（MSB）との間に小数点を持つ符号付き 1 6 ビットと、
- ・ 「FIX16_2」；第 1 3 ビットと第 1 4 ビットとの間に小数点を持つ符号付き 1 6 ビットと、
- ・ 「FIX32_1」；第 3 0 ビットと第 3 1 ビット（MSB）との間に小数点を持つ符号付き 3 2 ビットと、
- ・ 「FIX32_2」；第 2 9 ビットと第 3 0 ビットとの間に小数点を持つ符号付き 3 2 ビットと

が定義されているので、機械語命令置換部 1 2 2 は、例えば、

【 0 0 8 2 】

FIX16_1 a, b, c;

c = a * b;

というソースプログラムを、

fmulhh m0,Rc,Ra,Rb (固定小数点乗算命令)

という機械語命令に置き換える。

【 0 0 8 3 】

これによって、ユーザは、FIX16_1、FIX16_2、FIX32_1、FIX32_2という型について、通常のコンパイラにおける基本型と同様に宣言し、使用することができる。そして、生成された機械語命令は、周辺のコードと含めて、最適化部 1 3 0 における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

【 0 0 8 4 】

同様に、演算子定義ファイル 1 0 2 での定義により、異なる型間の暗黙の型変換のルール等が規定され（コンストラクタの定義による）、4 種類の S I M D 型、つまり、

- ・ 「VINT8x4」 ; 8 ビット整数データの 4 並列と、
- ・ 「VINT16x2」 ; 1 6 ビット整数データの 2 並列と、
- ・ 「VFIX161x2」 ; モード 0 (__ 1 系) の 1 6 ビット固定小数点データの 2 並列と、
- ・ 「VFIX162x2」 ; モード 1 (__ 2 系) の 1 6 ビット固定小数点データの 2 並列と

が定義されているので、機械語命令置換部 1 2 2 は、例えば、

【 0 0 8 5 】

VINT16x2 a, b, c;

c = a + b;

というソースプログラムを、

vaddh Rc,Ra,Rb (SIMD型加算命令)

という機械語命令に置き換える。

【 0 0 8 6 】

これによって、ユーザは、VINT8x2、VINT16x2、VFIX161x2、VFIX162x2という型について、通常のコンパイラにおける基本型と同様に宣言し、使用することができる。そして、生成された機械語命令は、周辺のコードと含めて、最適化部 1 3 0 における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

【 0 0 8 7 】

また、組み込み関数定義ファイル 1 0 3 においては、プロセッサ 1 が備える高機能命令を使用することが可能な関数（例えば、「_abs(a)」等）とそれに対応する高機能命令（例えば、1 つの機械語命令「abs Rb, Ra」等）が定義されているので、機械語命令置換部 1 2 2 は、例えば、

【 0 0 8 8 】

```
b = _abs(a);
```

というソースプログラムを、

```
abs Rb,Ra
```

という機械語命令に置き換える。

【 0 0 8 9 】

これによって、ユーザは、複雑な処理を、C++言語やアセンブラ命令で作成することなく、予め用意された組み込み関数を呼び出すだけで、実現することができる。そして、生成された機械語命令は、周辺のコードと含めて、最適化部 1 3 0 における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

【 0 0 9 0 】

同様に、また、組み込み関数定義ファイル 1 0 3 において、プロセッサ 1 が備える高機能命令を使用することが可能な関数（例えば、「_div(a, b)」等）とそれに対応する高機能命令（例えば、機械語命令列「extw, aslp, div」等）が定義されているので、機械語命令置換部 1 2 2 は、例えば、

【 0 0 9 1 】

```
c = _div(a, b);
```

というソースプログラムを、

```
extw    Mn,Rc,Ra
aslp    Mn,Rc,Mn,Rc,15
div     MHm,Rc,MHn,Rc,Rb
```

という機械語命令列に置き換える。

【 0 0 9 2 】

これによって、ユーザは、複雑な処理を、C++言語やアセンブラ命令で作成することなく、予め用意された組み込み関数を呼び出すだけで、実現することができる。そして、生成された機械語命令列は、周辺のコードと含めて、最適化部 130 における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

【 0 0 9 3 】

なお、組み込み関数定義ファイル 103 に列挙されている組み込み関数のうち、(1) 1つの機械語命令に変換される関数、(2) 2以上の機械語命令（機械語命令列）に変換される関数、及び、(3) レジスタ割り付けの対象とならない資源（アキュムレータなど）を指定することが可能な関数の代表的なもの（特に、メディア処理に有効なもの）は、以下の通りである。

【 0 0 9 4 】

(1) 1つの機械語命令に変換される組み込み関数

・「_bseq0(x)」:

入力の最上位ビットから何ビット0が連続するかを検出する関数である。書式は、以下の通りである。

```
int _bseq0(FIX16_1 val) // 0をカウント
int _bseq0(FIX16_2 val) // 0をカウント
int _bseq0(FIX32_1 val) // 0をカウント
int _bseq0(FIX32_2 val) // 0をカウント
```

これらの関数は、「val」中の連続する0の個数（ビット数）をカウントした値を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル 103 に定義されている通りである。

【 0 0 9 5 】

・ 「_bseq1(x)」 :

入力の最上位ビットから何ビット1が連続するかを検出する関数である。書式は、以下の通りである。

```
int _bseq1(FIX16_1 val) // 1をカウント
int _bseq1(FIX16_2 val) // 1をカウント
int _bseq1(FIX32_1 val) // 1をカウント
int _bseq1(FIX32_2 val) // 1をカウント
```

これらの関数は、「val」中の連続する1の個数（ビット数）をカウントした値が返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル 1 0 3 に定義されている通りである。

【 0 0 9 6 】

・ 「_bseq(x)」 :

入力の最上位ビットと同じ値が最上位ビットの次から何ビット連続するかを検出する関数である。書式は、以下の通りである。

```
int _bseq(FIX16_1 val)
int _bseq(FIX16_2 val)
int _bseq(FIX32_1 val)
int _bseq(FIX32_2 val)
```

これらの関数は、「val」の正規化ビット数を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル 1 0 3 に定義されている通りである。

【 0 0 9 7 】

・ 「_bcnt1(x)」 :

入力の全ビット中に何ビット1が含まれているかを検出する関数である。書式は、以下の通りである。

```
int _bcnt1(FIX16_1 val)
int _bcnt1(FIX16_2 val)
int _bcnt1(FIX32_1 val)
int _bcnt1(FIX32_2 val)
```

これらの関数は、「val」中の1の個数をカウントした値を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

【0098】

・「_extr(a,i1,i2)」:

入力 of 所定のビット位置を抽出して符号拡張する関数である。

関数である。書式は、以下の通りである。

int _extr (FIX16_1 val1, int val2, int val3)

int _extr (FIX16_2 val1, int val2, int val3)

int _extr (FIX32_1 val1, int val2, int val3)

int _extr (FIX32_2 val1, int val2, int val3)

これらの関数は、ビット位置val2からビット位置val3で示されたval1のビットフィールドを抽出し、符号拡張を行った結果を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

【0099】

・「_extru(a,i1,i2)」:

入力 of 所定のビット位置を抽出してゼロ拡張する関数である。書式は、以下の通りである。

unsigned int _extru (FIX16_1 val, int val2, int val3)

unsigned int _extru (FIX16_2 val, int val2, int val3)

unsigned int _extru (FIX32_1 val, int val2, int val3)

unsigned int _extru (FIX32_2 val, int val2, int val3)

これらの関数は、ビット位置val2からビット位置val3で示されたval1のビットフィールドを抽出し、ゼロ拡張を行った結果を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

【0100】

(2) 2以上の機械語命令(機械語命令列)に変換される組み込み関数

・「_modulo_add()」:

モジュロアドレッシングのアドレス更新を行う関数である。書式は、以下の通

りである。

```
_modulo_add(void *addr, int imm, int mask, size_t size, void *base)
```

ここで、各引数の意味は以下の通りである。

addr: 更新前のアドレス又はアドレス下位(モジュロ部分)

imm: 加算値 (データ数)

mask: マスク幅 (モジュロ幅)

size: データのサイズ (2のべき乗)

base: ベースアドレス (配列の先頭アドレス)

この関数は、モジュロアドレッシングにより、アドレスaddrから加算値immだけ加算した結果を返す。

【 0 1 0 1 】

この関数に対応する機械語命令は、組み込み関数定義ファイル 1 0 3 に定義されている通りである。つまり、モジュロアドレッシング計算のため、第一の入力の所定のビットフィールドを第二の入力の所定のビットフィールドで置き換える命令(addmsk)を活用している関数である。使用例は、以下の通りである。

```
int array[MODULO];
p = array;
for (i = 0; i < 100; i++) {
    *q++ = *p;
    p = (int *)_modulo_add(p, 1, N, sizeof(int), array);
}
```

ここで、変数MODULOは2のべき乗 (2^N) である。この使用例では、配列arrayの100個の要素を MODULO*SIZEバイトアライメントして配置している。

【 0 1 0 2 】

・ 「_brev_add()」 :

ビットリバースアドレッシングのアドレス更新を行う関数である。書式は、以下の通りである。

```
_brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
```

ここで、各引数の意味は以下の通りである。

addr: 更新前のアドレス

cnt: ビットリバースカウンタ

mm: 加算値 (データ数)

mask: マスク幅 (リバース幅)

size: データのサイズ ※ 2のべき乗

base: ベースアドレス (配列の先頭アドレス)

この関数は、ビットリバースアドレッシングにより、ビットリバースカウンタ cnt に対応するアドレス addr から加算値 imm だけ加算した結果を返す。

【 0 1 0 3 】

この関数に対応する機械語命令は、組み込み関数定義ファイル 1 0 3 に定義されている通りである。つまり、ビットリバースアドレッシング計算のため、第一の入力の所定のビットフィールドに対してビット単位の位置反転を行なう命令 (mskbrvh) を活用している関数である。使用例は、以下の通りである。

```
int array[BREV];
p = array;
for (i = 0; i < 100; i++) {
    *q++ = *p;
    p = (int *)_brev_add(p, i, 1, N, sizeof(int), array);
}
```

ここで、変数 BREV は 2 のべき乗 (2^N) である。この使用例では、配列 array の 1 0 0 個の要素を BREV*SIZE バイトアライメントして配置する。

【 0 1 0 4 】

(3) レジスタ割り付けの対象とならない資源 (アキュムレータなど) を指定することが可能な関数

組み込み関数定義ファイル 1 0 3 には、最適化におけるレジスタ割り付けの対象となる資源である汎用レジスタに加えて、レジスタ割り付けの対象とならない資源である (暗黙の資源である) アキュムレータも更新する演算 (乗算、積和) であって、アキュムレータを参照型として一時変数を指定することが可能な組み込

み関数（乗算「_mul」、積和「_mac」）が用意されている。具体的な書式は、それぞれ、以下の通りである。

【 0 1 0 5 】

```
_mul(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
```

この関数は、変数 a 変数 b とを乗算し、結果の 6 4 bit データの上位 3 2 ビットを乗算用上位アキュムレータ MH に、下位 3 2 ビットを乗算用下位アキュムレータ ML に設定し、さらに、アキュムレータ MH の下位 1 6 ビットとアキュムレータ ML の上位 1 6 ビットとを連結した 3 2 ビットデータを変数 c にも設定する。

```
_mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
```

この関数は、変数 a と変数 b とを乗算した結果に、乗算用上位アキュムレータ MH と乗算用下位アキュムレータ ML とを連結した 6 4 ビットデータを加算し、結果の 6 4 bit データの上位 3 2 ビットを乗算用上位アキュムレータ MH に、下位 3 2 ビットを乗算用下位アキュムレータ ML に設定し、さらに、乗算用上位アキュムレータの下位 1 6 ビットと乗算用下位アキュムレータ ML の上位 1 6 ビットとを連結した 3 2 ビットデータを変数 c にも設定する。

【 0 1 0 6 】

使用例は、以下の通りである。

```
long mh,ml;
```

```
_mul(mh,ml,dummy,a,b);
```

```
_mac(mh,ml,e,c,d);
```

というソースプログラムに対して、機械語命令置換部 1 2 2 は、組み込み関数定義ファイル 1 0 3 での定義に従って、

```
mul m0,Rx,Ra,Rb
```

```
mov r0,mh0
```

```
mov r1,mh1
```

```
mov mh0,r0
```

```
mov mh1,r1
```

```
mac m0,Re,Rc,Rd,m0
```


という機械語命令列に置き換える。なお、上記機械語命令列のうち、第 1 ～ 第 3 行が関数 `_mul` に対応するものであり、第 4 ～ 第 6 行が関数 `_mac` に対応するものである。このような機械語命令列は、最適化部 1 3 0 において、冗長除去により、第 2 ～ 第 5 行が削除され、

```
mul m0,Rx,Ra,Rb
```

```
mac m0,Re,Rc,Rd,m0
```

という機械語命令列に最適化される。このように、レジスタ割り付けの対象としない資源（アキュムレータなど）を指定することが可能な組み込み関数を使用された場合には、その資源に対する定義（値の格納）と使用（値の参照）との組がコンパイラ内部の最適化（最適化部 1 3 0）にて削除される可能性が高いため、このような組み込み関数は、最適化の点においても有効である。

【 0 1 0 7 】

次に、本コンパイラ 1 0 0 の特徴的な動作のうち、最適化部 1 3 0 による動作を説明する。

図 7 4 は、最適化部 1 3 0 の引数最適化部 1 3 1 の動作を示すフローチャートである。引数最適化部 1 3 1 は、組み込み関数（`extr`、`extru` など）の引数に応じて、適切な命令又はシーケンス（アルゴリズム）を生成するために、その関数の引数がすべて定数の場合には（ステップ S 1 1 0 で左）、それら定数を畳み込んで得られる定数値をオペランドとする機械語命令を生成し（ステップ S 1 1 1）、引数の一部が定数の場合には（ステップ S 1 1 0 で中央）、即値オペランドの機械語命令を生成し（ステップ S 1 1 2）、引数のすべてが変数の場合には（ステップ S 1 1 0 で右）、レジスタオペランドの命令列を生成する（ステップ S 1 1 3）。

【 0 1 0 8 】

例えば、

```
d = _extru(0xffff, 7, 4);
```

のように、引数のすべて定数の場合には、

```
movRd,0xf
```

のように、畳み込んで定数値を設定した機械語命令を生成する。

一方、

```
d = _extru(a, 7, 4);
```

のように、一部の引数が定数の場合には、

```
extruRd,Ra,7,4
```

のように、即値オペランドの命令を生成する。

さらに一方、

```
d = _extru(a, b, c);
```

のように、すべての引数が変数の場合には、

```
aslRe,Rb,8
```

```
andRf,Rc,0x1f
```

```
orRg,Re,Rf
```

```
extruRd,Ra,Rg
```

のように、レジスタオペランドの命令列を生成する。

【0 1 0 9】

このように、引数最適化部 1 3 1 により、1 つの組み込み関数は、常に同じ機械語命令が固定的に生成されるのではなく、その引数の性質に応じて最適化された機械語命令（又は機械語命令列）が生成されることになる。

【0 1 1 0】

図 7 5 は、最適化部 1 3 0 の型変換最適化部 1 3 2 の動作を説明するための演算木を示す図である。型変換最適化部 1 3 2 は、異なる型間の演算を効率化するために、ソースプログラム中の一定表記の演算に対して、型変換を伴う機械語命令（`fmulhw`など）を生成する。

【0 1 1 1】

通常の C 言語では、16bit x 16bit の結果の型は 16bit である。16bit x 16bit -> 32bit の命令は存在するが、次のように、2 つの機械語命令が生成されてしまう。例えば、

```
f32 = f16 * f16;
```

の記述に対して、

```
fmulhh // 16bit x 16bit -> 16bit
```

asl // 16bit -> 32bitの型変換

の2命令が生成されてしまう。

【0 1 1 2】

そこで、型変換最適化部 1 3 2 は、ソースプログラムにおいて、(FIX32)16bit * (FIX32)16bit と記述されている場合には、通常であれば、図 7 5 (a) に示される演算木が生成されてしまう（型変換のコードが生成されてしまう）ところ、この演算木を図 7 5 (b) に示される演算木に変換することにより、16bit x 16bit --> 32bitの1命令 (fmulhw) を生成する。

【0 1 1 3】

図 7 6 は、レイテンシ最適化部 1 3 3 の動作を説明するためのサンプルプログラムの例を示す図である。レイテンシ最適化部 1 3 3 は、ソースプログラム 1 0 1 に組み込まれたアセンブラ命令（最適化asm文）でのレイテンシに関する指示（サイクル数の指定）に基づいて、特定の区間あるいは特定の動作において、指定されたサイクル数だけ実行時間がかかるように、機械語命令をスケジューリングする。

【0 1 1 4】

ユーザは、2種類の指定方法によって、レイテンシを設定することができる。

その1つの方法は、図 7 6 (a) に示されるプログラムにおける指定 (LATENCY L1, L2, 2;) のように、特定の命令に付けられたラベル間のレイテンシを指定する方法である。図 7 6 (a) の例では、レイテンシ最適化部 1 3 3 は、プロセッサ 1 によって命令wteが実行されてから命令rdeが実行されるまでに2サイクルだけ経過するように、機械語命令列の配置をスケジューリングする。

【0 1 1 5】

他の1つの方法は、図 7 6 (b) に示されるプログラムにおける指定 (命令wte内におけるLATENCY(2)) のように、拡張レジスタ部 8 0 にアクセスする命令 (rd, wt, rde, wte) について、次に拡張レジスタ部 8 0 にアクセスするまでのレイテンシを指定する方法である。図 7 6 (b) の例では、レイテンシ最適化部 1 3 3 は、プロセッサ 1 によって命令wteが実行され、拡張レジスタ部 8 0 にアクセスされてから再び拡張レジスタ部 8 0 にアクセスされるまでに2サイクルだけ

経過するように、機械語命令列の配置をスケジューリングする。

【0 1 1 6】

このようなレイテンシの設定によって、inline展開先のコードとの間で最適化(命令結合、冗長除去、命令並べ替え、レジスタ割り付け)が可能になるとともに、指定した命令間又はアクセス間でのレイテンシが確保される。つまり、従来であれば、ユーザは、明示的にnop命令を挿入しなければならないところ、このコンパイラ100によれば、必要な命令又はアクセスについて必要なレイテンシを指定するだけで済む。

【0 1 1 7】

図77は、パーサ一部110の固定小数点モード切替部111の動作を説明するための図である。

固定小数点モード切替部111は、ソースプログラム101中に、固定小数点モードを退避・復帰させるプリAGMA指令(例えば、`#pragma _save_fxpmode func`等)を検出すると、プロセッサ1のPSR31のビットFXPの退避・復帰を行う機械語命令を生成する。

【0 1 1 8】

なお、前提となる固定小数点に関する仕様として、_1系(FIX16_1, FIX32_1)と_2系(FIX16_2, FIX32_2)の型が存在し、ハードウェア(プロセッサ1)では、PSR31の1ビット(FXP)でモード切り替えをしており、さらに、関数内には単一の系統しか使用できないという条件が存在する。

【0 1 1 9】

そこで、これら2つの体系をプログラム上で切り替えて使用方法として、他の体系からcallされる可能性のある関数にプリAGMA(`#pragma _save_fxpmode 関数名`)を指定することとしている。これによって、固定小数点モード切替部111は、当該関数の先頭と末尾にFIX型モードの退避・復帰に相当するコードを挿入する。また、各関数のFIX型宣言をサーチし、その関数をいずれのFIX型モードでコンパイルするかを決定し、モード設定のコードを挿入する。

【0 1 2 0】

図77(a)は、そのプリAGMA指令を伴う関数の例を示している。図77(a)

）の右側に記されたコメントが固定小数点モード切替部 1 1 1 による挿入処理であり、その具体的な処理は、図 7 7 （b）に示される通りである。

【0 1 2 1】

このようなプラグマ指令の適用例は、図 7 7 （c）に示される通りである。例えば、4 つの関数、f11, f21, f22, f23 について、関数 f11: _1 系 は 関数 f21: _2 系を call し、関数 f21: _2 系 は 関数 f22: _2 系を call し、関数 f22: _2 系 は 関数 f23: _2 系を call する場合、他のモードから call される可能性がある関数は f21 のみなので、この関数についてのみプラグマ指定するだけで、正常なモード切り替えが可能となる。

【0 1 2 2】

以上のように、本実施の形態におけるコンパイラ 1 0 0 によれば、演算子定義ファイル 1 0 2 及び組み込み関数定義ファイル 1 0 3 と機械語命令置換部 1 2 2 との連携処理により、ユーザは、モード 0 及び 1 の固定小数点型を通常の型として宣言し、使用することができるとともに、高級言語のレベルで組み込み関数を呼び出すことで、プロセッサ 1 が備える高機能な機械語命令を効率的に生成させることができる。

【0 1 2 3】

また、引数最適化部 1 3 1 による組み込み関数の引数についての最適化により、効率的なオペランドを持つ機械語命令が生成される。また、型変換最適化部 1 3 2 による型変換についての最適化により、型変換を伴う演算は、プロセッサ 1 が備える高機能な 1 つの機械語命令に変換される。さらに、レイテンシ最適化部 1 3 3 による機械語命令のスケジューリングにより、ユーザは、nop 命令を挿入することなく、特定の命令間あるいは拡張レジスタへのアクセスにおけるレイテンシを確保することができる。

【0 1 2 4】

以上、本発明に係るコンパイラについて、実施の形態に基づいて説明したが、本発明は、この実施の形態に限定されるものではない。

例えば、本実施の形態では、固定小数点の型は、1 6 ビット又は 3 2 ビットで、かつ、小数点 が MSB 又はその下位桁に位置したが、このようなフォーマットだ

けに限られず、8ビットや64ビットの固定小数点や、他の桁に小数点が位置する型を対象としてもよい。

【0125】

また、ユーザに対する開発支援ツールとして、クラスライブラリを活用した動作検証手法を提供してもよい。つまり、通常であれば、図78(a)に示されるように、本実施の形態におけるターゲットマシン（プロセッサ1）用のクロスコンパイラ（コンパイラ100）を用いてテストソースと定義ファイル102及び103等をコンパイルし、得られたプロセッサ1用の機械語プログラムを専用のシミュレータで実行させることによって動作検証を行うが、これに代えて、図78(b)に示されるように、開発用ホストマシン（例えば、米国インテル社製のプロセッサ）を対象としたクラスライブラリ（演算子定義ファイル102及び組み込み関数定義ファイル103それぞれをプロセッサ1ではなく、ホストマシンの機械語命令に対応させる定義ファイル）を用意し、テストソース、定義ファイル102及び103と共に、ネイティブコンパイラ（Visual C++（R）等）でコンパイルし、得られた機械語プログラムをそのままホストマシンで実行させてもよい。これによって、慣れた環境で高速にシミュレーションし、動作検証を行うことが可能となる。

【0126】

さらに、本実施の形態では、ターゲットプロセッサに特有の機械語命令に対応する演算子や組み込み関数は、ヘッダファイル（定義ファイル102及び103）として供給されたが、本発明に係るコンパイラは、このような定義ファイルでの情報をコンパイラ自身の中に組み込んだ構成としてもよい。つまり、本発明に係るコンパイラは、上記定義ファイル102及び103が組み込まれた一体形式のプログラムであって、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、ソースプログラムを解析するパーサステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、中間コード変換ステップは、前記中間コードのうち、前記演算処理定義情報で定

義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う構成としてもよい。これによって、ユーザは、ソースプログラムにおいて定義ファイルをインクルードする必要がなくなる。

【 0 1 2 7 】

【発明の効果】

以上の説明から明らかなように、本発明に係るコンパイラ用プログラムは、ソースプログラムを機械語プログラムに翻訳するコンパイラ用のプログラムであって、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、前記プログラムは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記演算処理定義情報で定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

【 0 1 2 8 】

例えば、本発明に係るプログラムは、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、データとメソッドとからなるクラスが定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステッ

ブとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義されたクラスを参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うプログラムである。

【 0 1 2 9 】

これによって、ソースプログラムにおいて、ヘッダファイルで定義されているクラスを参照しているステートメントがある場合には、そのステートメントに対応する中間コードは、機械語命令に変換された後に最適化処理の対象となるので、周辺の機械語命令と含めて最適化が施され得る。また、コンパイラは、コンパイラ自身の機能（最適化処理）だけでなく、ヘッダファイルでの定義との連携によって最適化を図っているので、ヘッダファイルの内容を更新するだけで、最適化の対象となるステートメントを増加させたり、最適化のレベルを向上させたりすることができる。

【 0 1 3 0 】

ここで、前記クラスは、固定小数点型を定義するクラスであり、前記検出サブステップでは、前記固定小数点型のデータを使用している中間コードを検出してもよい。そして、前記クラスのメソッドは、前記固定小数点型のデータを対象とする演算子を定義するメソッドであり、前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換えてもよい。

【 0 1 3 1 】

これによって、ヘッダファイルで定義されている固定小数点型及び演算子についても、通常の型と同様に、コンパイラによって対応する中間コードや機械語命令に変換されるので、ユーザは、ソースプログラムにおいてそのヘッダファイルをインクルードしておくだけで、通用の型と同様に、ターゲットプロセッサ特有の固定小数点モードに対応した型の宣言や使用をすることが可能となる。

【 0 1 3 2 】

また、本発明に係るコンパイラ用プログラムは、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、関数が定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義された関数を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

【 0 1 3 3 】

これによって、ソースプログラムにおいて、ヘッダファイルで定義されている関数（組み込み関数）を参照しているステートメントがある場合には、そのステートメントに対応する中間コードは、そのヘッダファイルで定義されている機械語命令に変換された後に最適化処理の対象となるので、周辺の機械語命令と含めて最適化が施され得る。また、ユーザは、プロセッサ専用の高機能命令を使用したい場合には、ソースプログラムにおいてそのヘッダファイルをインクルードしておくとともに、必要な組み込み関数を呼び出す旨の記述をしておくだけで済む。つまり、アセンブラ命令でコーディングすることから解放される。

【 0 1 3 4 】

以上のように、本発明に係るコンパイラにより、ターゲットプロセッサが備える高機能な専用命令が効率的に生成され、高いレベルで最適化が施されるとともに、ヘッダファイルによる機能拡張等の柔軟な対応が可能となり、特に、コードサイズと実行速度の両面で厳しい仕様が求められるメディア処理アプリケーションの開発ツールとして、その実用的価値は極めて高い。

【図面の簡単な説明】

【図 1】 本発明に係るコンパイラの対象となるプロセッサの概略ブロック図である。

【図 2】 同プロセッサの算術論理・比較演算器の概略図を示す。

【図 3】 同プロセッサのバレルシタの構成を示すブロック図である。

【図 4】 同プロセッサの変換器の構成を示すブロック図である。

【図 5】 同プロセッサの除算器の構成を示すブロック図である。

【図 6】 同プロセッサの乗算・積和演算器の構成を示すブロック図である。

【図 7】 同プロセッサの命令制御部の構成を示すブロック図である。

【図 8】 同プロセッサの汎用レジスタ（R0～R31）の構造を示す図である。

【図 9】 同プロセッサのリンクレジスタ（LR）の構造を示す図である。

【図 10】 同プロセッサの分岐レジスタ（TAR）の構造を示す図である。

【図 11】 同プロセッサのプログラム状態レジスタ（PSR）の構造を示す図である。

【図 12】 同プロセッサの条件フラグレジスタ（CFR）の構造を示す図である。

【図 13】 同プロセッサのアキュムレータ（M0，M1）の構造を示す図である。

【図 14】 同プロセッサのプログラムカウンタ（PC）の構造を示す図である。

【図 15】 同プロセッサのPC退避用レジスタ（IPC）の構造を示す図である。

【図 16】 同プロセッサのPSR退避用レジスタ（IPSR）の構造を示す図である。

【図 17】 同プロセッサのパイプライン動作を示すタイミング図である。

【図 18】 同プロセッサによる命令実行時の各パイプライン動作を示すタ

イメージ図である。

【図 1 9】 同プロセッサの並列動作を示す図である。

【図 2 0】 同プロセッサが実行する命令のフォーマットを示す図である。

【図 2 1】 カテゴリー「ALUadd（加算）系」に属する命令を説明する図である。

【図 2 2】 カテゴリー「ALUsub（減算）系」に属する命令を説明する図である。

【図 2 3】 カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図である。

【図 2 4】 カテゴリー「CMP（比較演算）系」に属する命令を説明する図である。

【図 2 5】 カテゴリー「mul（乗算）系」に属する命令を説明する図である。

【図 2 6】 カテゴリー「mac（積和演算）系」に属する命令を説明する図である。

【図 2 7】 カテゴリー「msu（積差演算）系」に属する命令を説明する図である。

【図 2 8】 カテゴリー「MEMld（メモリ読み出し）系」に属する命令を説明する図である。

【図 2 9】 カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図である。

【図 3 0】 カテゴリー「BRA（分岐）系」に属する命令を説明する図である。

【図 3 1】 カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図である。

【図 3 2】 カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図である。

【図 3 3】 カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図である。

【図 3 4】 カテゴリー「CNV（一般変換）系」に属する命令を説明する図である。

【図 3 5】 カテゴリー「SATvlpk（飽和处理）系」に属する命令を説明する図である。

【図 3 6】 カテゴリー「ETC（その他）系」に属する命令を説明する図である。

【図 3 7】 本発明に係るコンパイラの構成を示す機能ブロック図である。

【図 3 8】 演算子定義ファイルのリストの一部を示す図である。

【図 3 9】 演算子定義ファイルのリストの一部を示す図である。

【図 4 0】 演算子定義ファイルのリストの一部を示す図である。

【図 4 1】 演算子定義ファイルのリストの一部を示す図である。

【図 4 2】 演算子定義ファイルのリストの一部を示す図である。

【図 4 3】 演算子定義ファイルのリストの一部を示す図である。

【図 4 4】 演算子定義ファイルのリストの一部を示す図である。

【図 4 5】 演算子定義ファイルのリストの一部を示す図である。

【図 4 6】 演算子定義ファイルのリストの一部を示す図である。

【図 4 7】 演算子定義ファイルのリストの一部を示す図である。

【図 4 8】 演算子定義ファイルのリストの一部を示す図である。

【図 4 9】 演算子定義ファイルのリストの一部を示す図である。

【図 5 0】 演算子定義ファイルのリストの一部を示す図である。

【図 5 1】 演算子定義ファイルのリストの一部を示す図である。

【図 5 2】 演算子定義ファイルのリストの一部を示す図である。

【図 5 3】 演算子定義ファイルのリストの一部を示す図である。

【図 5 4】 演算子定義ファイルのリストの一部を示す図である。

【図 5 5】 演算子定義ファイルのリストの一部を示す図である。

【図 5 6】 演算子定義ファイルのリストの一部を示す図である。

【図 5 7】 演算子定義ファイルのリストの一部を示す図である。

【図 5 8】 演算子定義ファイルのリストの一部を示す図である。

【図 5 9】 演算子定義ファイルのリストの一部を示す図である。

- 【図 6 0】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 1】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 2】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 3】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 4】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 5】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 6】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 7】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 8】 演算子定義ファイルのリストの一部を示す図である。
- 【図 6 9】 組み込み関数定義ファイルのリストの一部を示す図である。
- 【図 7 0】 組み込み関数定義ファイルのリストの一部を示す図である。
- 【図 7 1】 組み込み関数定義ファイルのリストの一部を示す図である。
- 【図 7 2】 組み込み関数定義ファイルのリストの一部を示す図である。
- 【図 7 3】 機械語命令置換部の動作を示すフローチャートである。
- 【図 7 4】 最適化部の引数最適化部の動作を示すフローチャートである。
- 【図 7 5】 最適化部の型変換最適化部の動作を説明するための演算木を示す図である。

【図 7 6】 レイテンシ最適化部の動作を説明するためのサンプルプログラムの例を示す図である。

【図 7 7】 パーサー部の固定小数点モード切替部の動作を説明するための図である。

【図 7 8】 クラスライブラリを活用した動作検証手法を説明するための図である。

【符号の説明】

- 1 プロセッサ
- 1 0 命令制御部
- 2 0 デコード部
- 3 0 レジスタファイル
- 3 1 プログラム状態レジスタ (P S R)

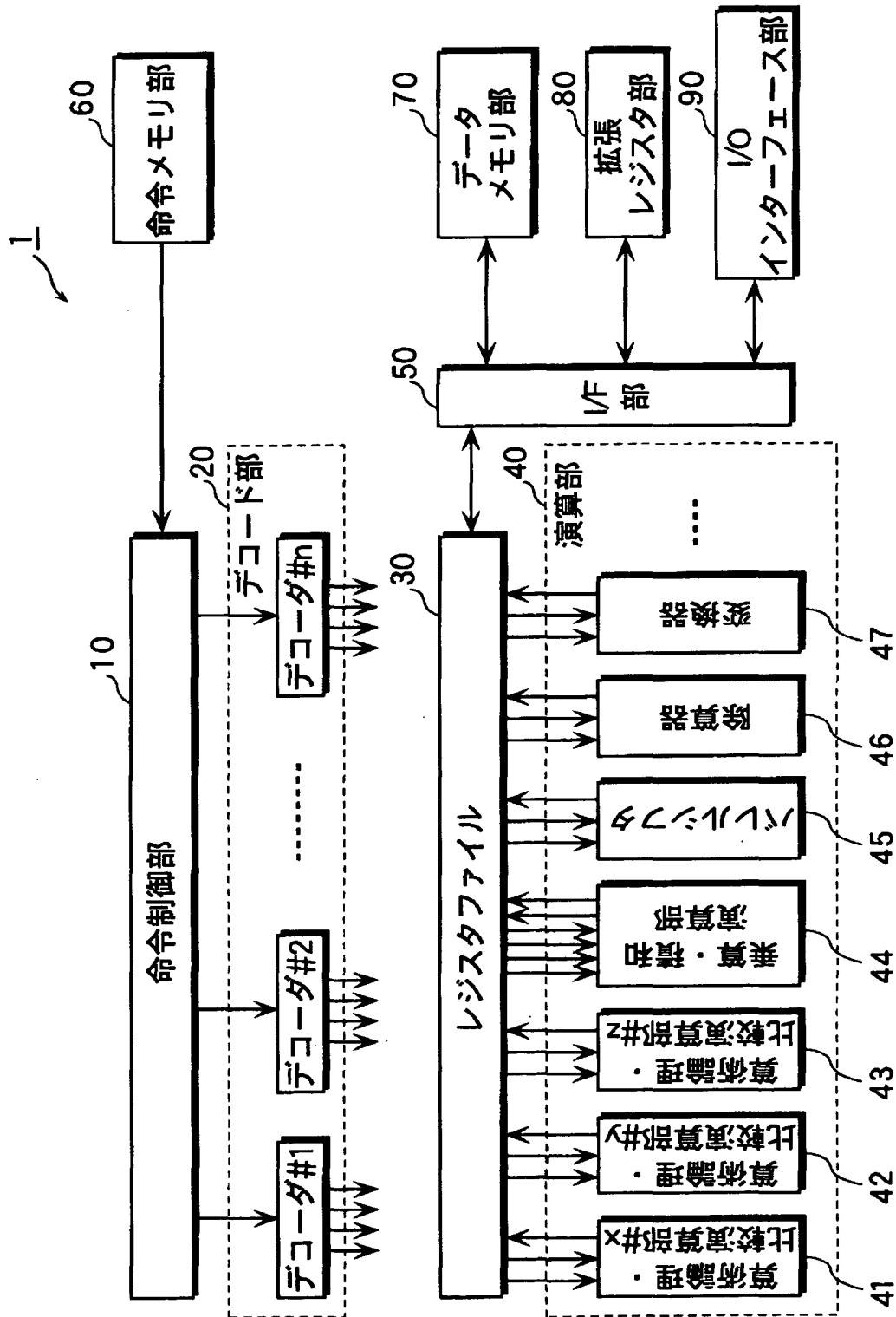
- 32 条件フラグレジスタ (CFR)
- 33 プログラムカウンタ (PC)
- 34 PC退避用レジスタ (IPC)
- 35 PSR退避用レジスタ (IPSR)
- 40 演算部
- 41～43 算術論理・比較演算器
- 44 積和演算器
- 45 バレルシフタ
- 46 除算器
- 47 変換器
- 50 I/F部
- 60 命令メモリ部
- 70 データメモリ部
- 80 拡張レジスタ部
- 90 I/Oインターフェース部
- 100 コンパイラ
- 101 ソースプログラム
- 102 演算子定義ファイル
- 103 組み込み関数定義ファイル
- 105 機械語プログラム
- 110 パーサー部
- 111 固定小数点モード切替部
- 120 中間コード変換部
- 121 中間コード生成部
- 122 機械語命令置換部
- 122 a 置換テーブル
- 130 最適化部
- 131 引数最適化部
- 132 型変換最適化部

- 1 3 3 レイテンシ最適化部
- 1 4 0 コード生成部

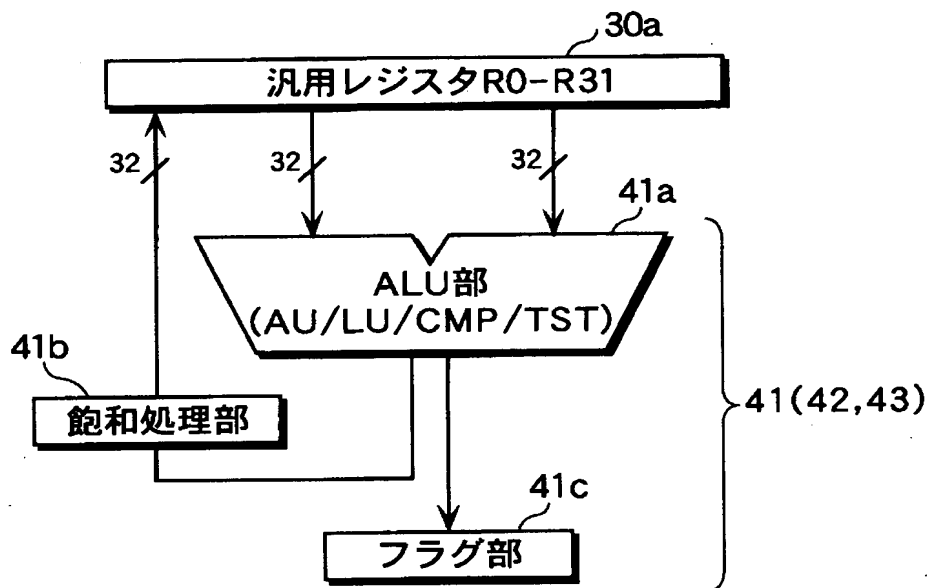
【書類名】

図面

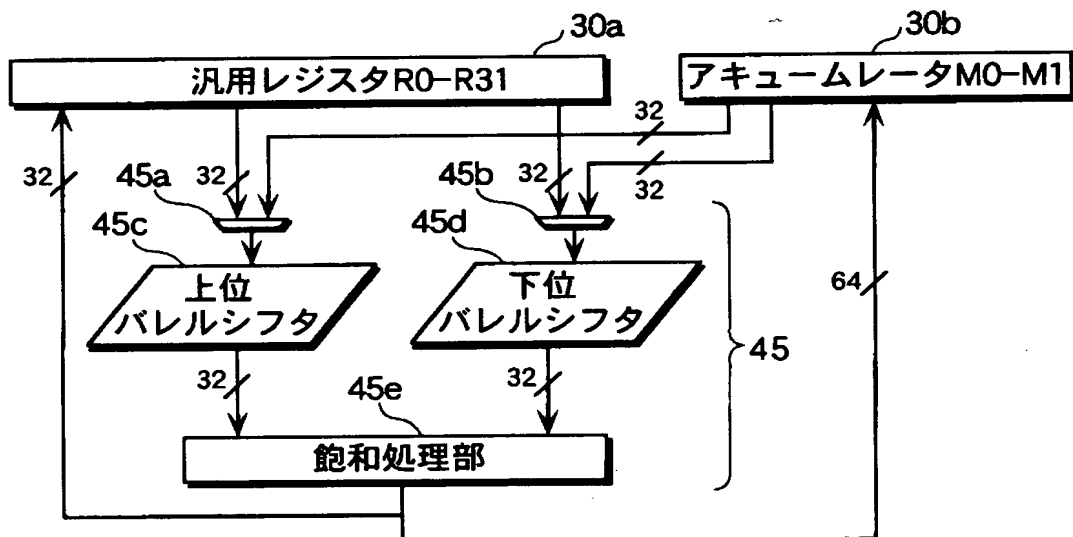
【図 1】



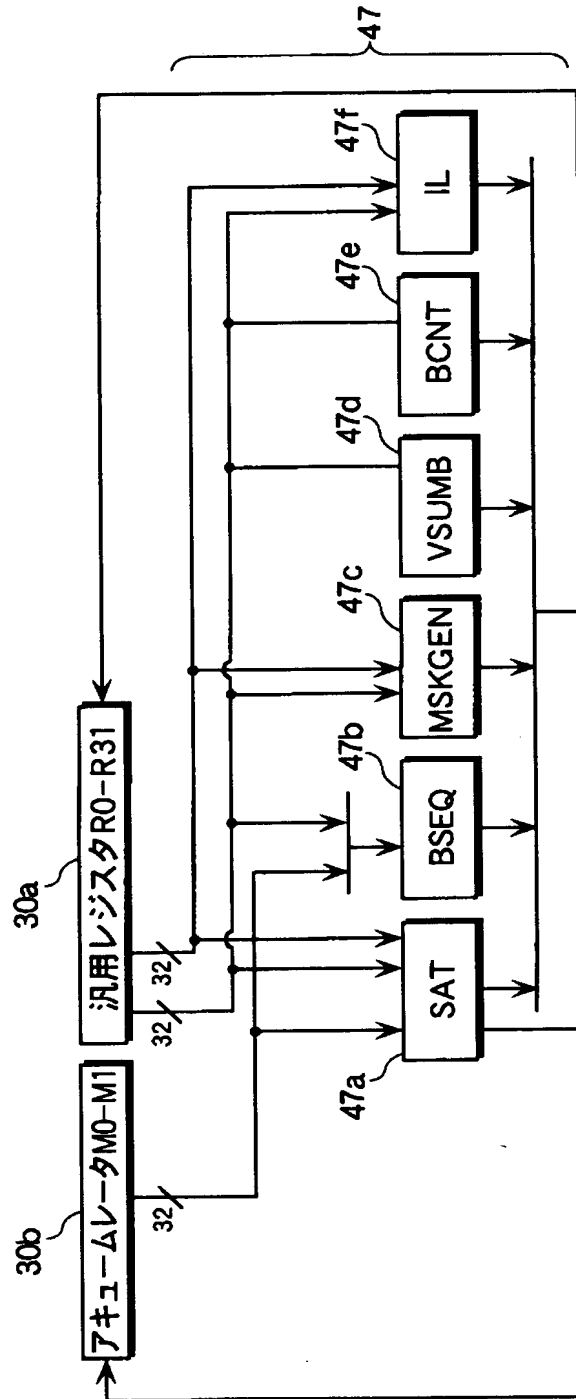
【図 2】



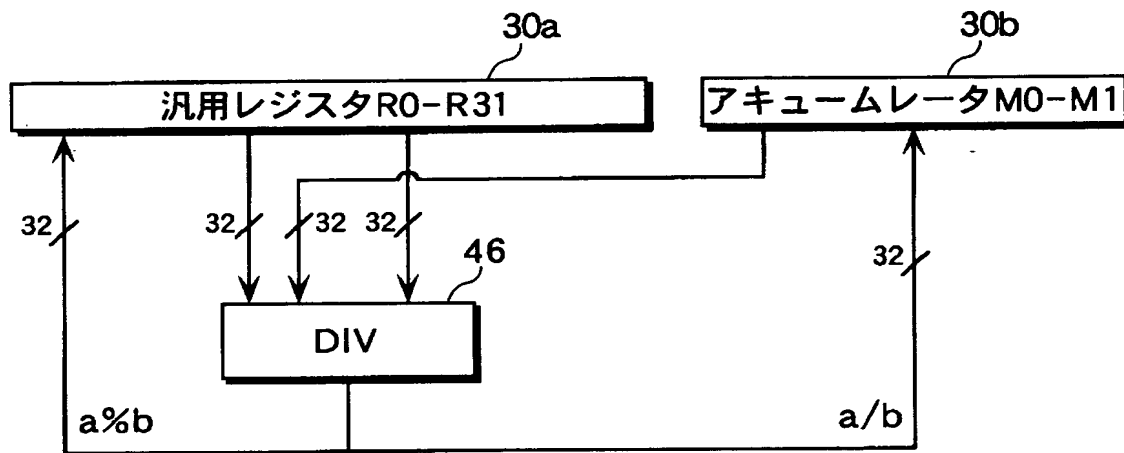
【図 3】



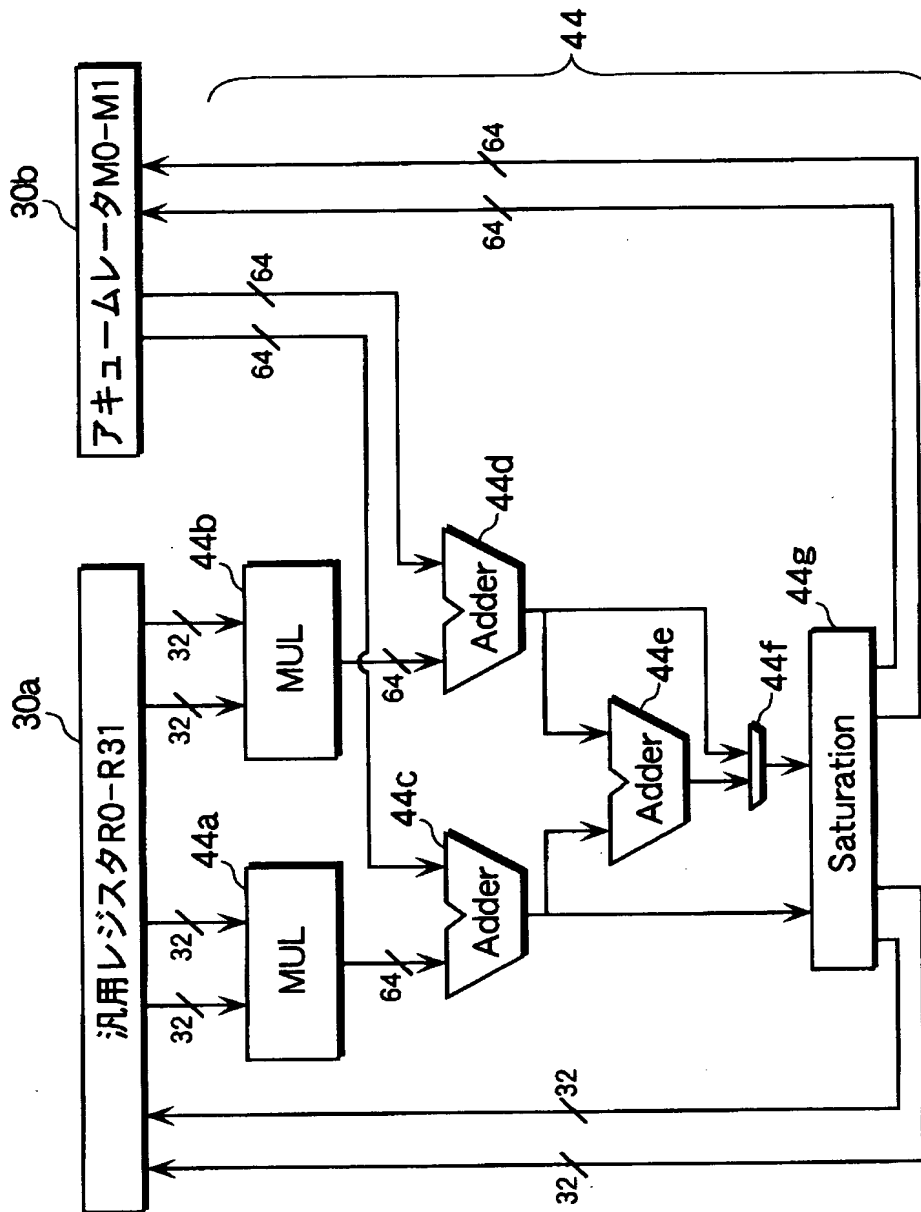
【図 4】



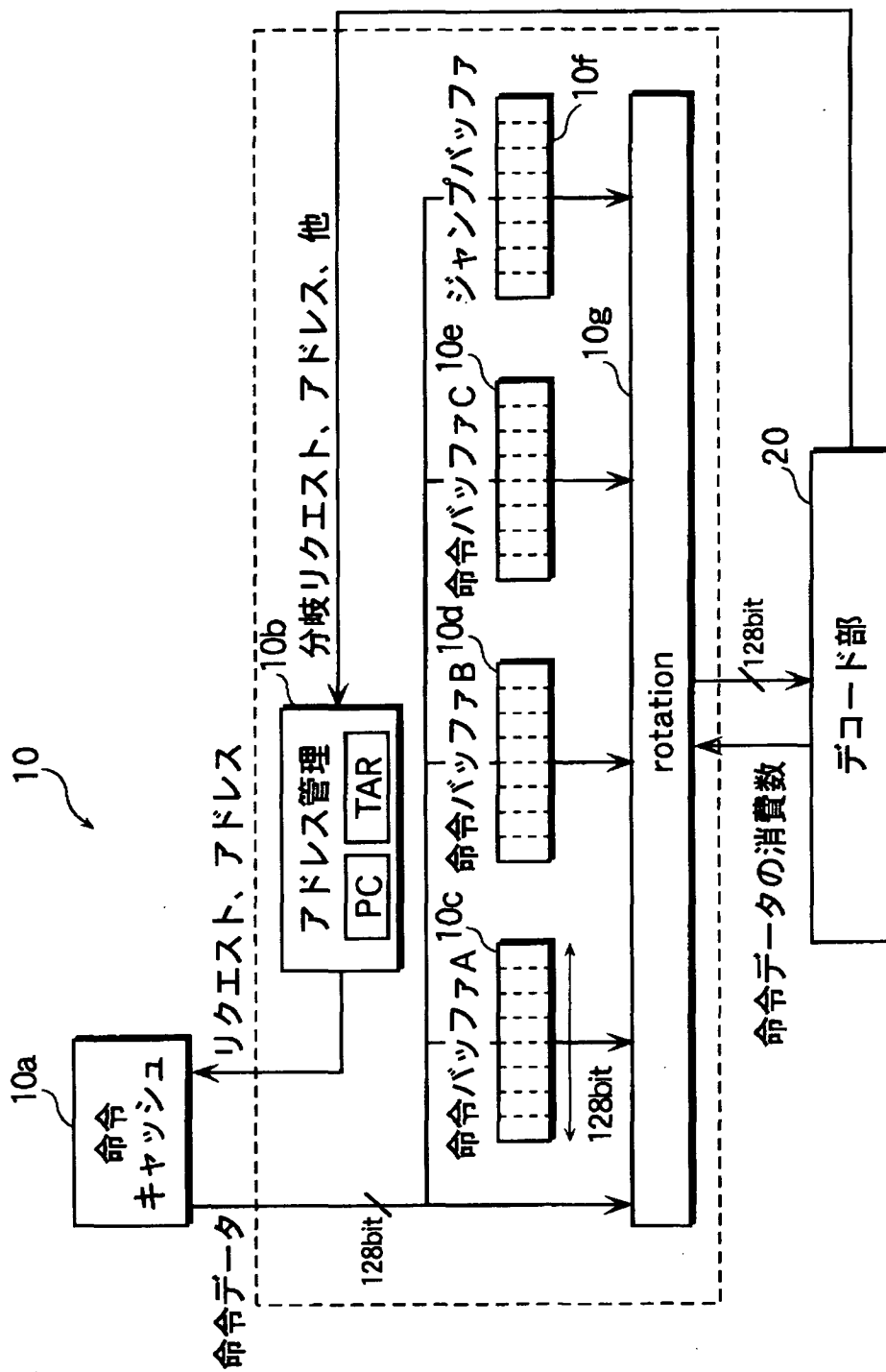
【図 5】



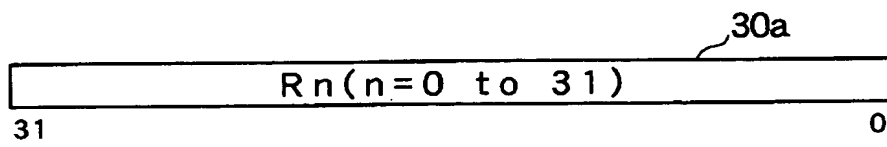
【図 6】



【図 7】



【図 8】



【図 9】



【図 10】



【図 1 1】

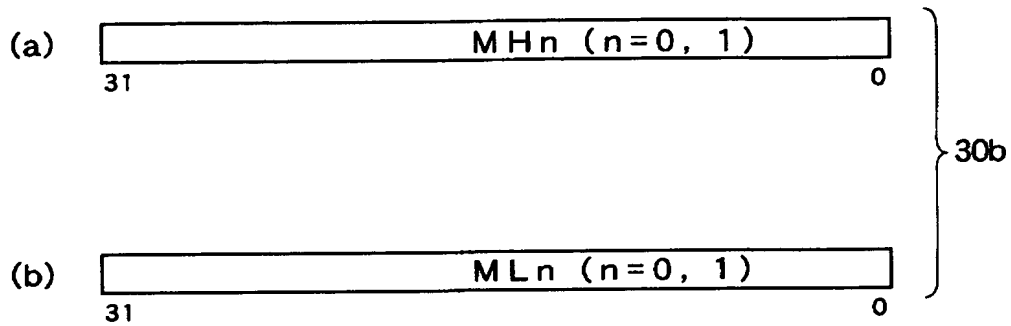
ビット	31	30	29	28	27	26	25	24
ビット名	reserved	SWE	FXP	reserved	IH	EH	PL	
ビット	23	22	21	20	19	18	17	16
ビット名	LPIE3	LPIE2	LPIE1	LPIE0	reserved	reserved	AEE	IE
ビット	15	14	13	12	11	10	9	8
ビット名	Reserved							
ビット	7	6	5	4	3	2	1	0
ビット名	IM[7:0]							

【図 12】

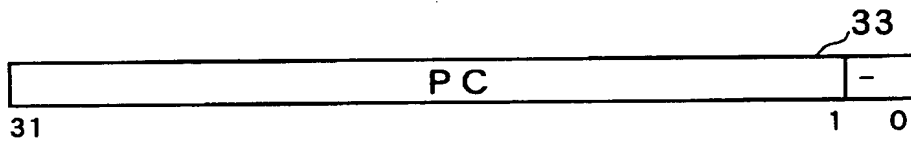
32

ビット	31	30	29	28	27	26	25	24
ビット名	ALN		reserved	BPO				
ビット	23	22	21	20	19	18	17	16
ビット名	reserved				VC3	VC2	VC1	VC0
ビット	15	14	13	12	11	10	9	8
ビット名	reserved						OVS	CAS
ビット	7	6	5	4	3	2	1	0
ビット名	C7	C6	C5	C4	C3	C2	C1	C0

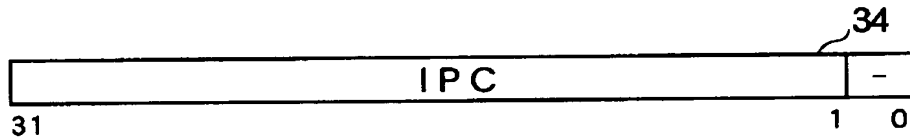
【図 1 3】



【図 1 4】



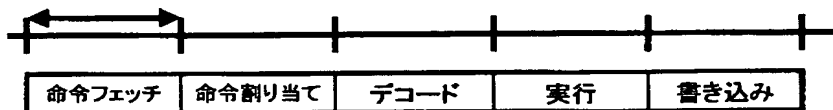
【図 1 5】



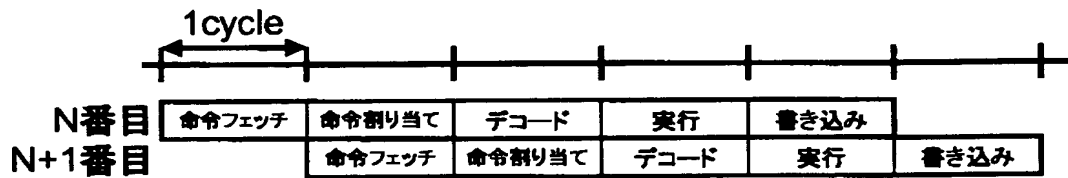
【図 1 6】



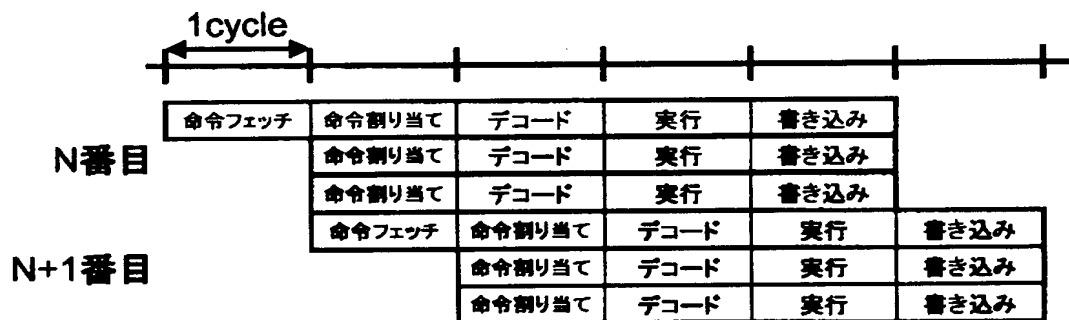
【図 1 7】



【図 18】

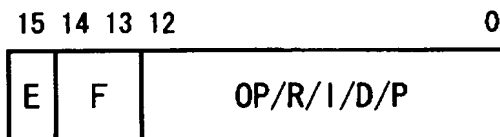


【図 19】

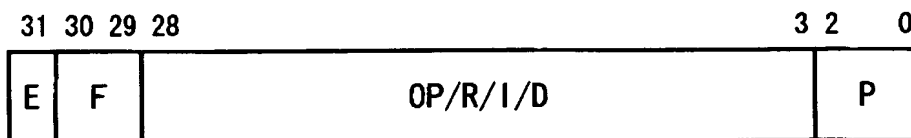


【図 20】

(a) 16ビット命令フォーマット



(b) 32ビット命令フォーマット




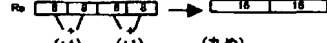
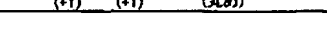
【図 21】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31
ALU add 系	S I M D	ワード	add	Rc, Ra, Rb Rb, Ra, i12s SP, i19s Ra2, Rb2 Rc3, Ra3, Rb3 Ra2, i05s SP, i11s					32
			addu	Rb, GP, i18u Rb, SP, i18u Ra3, SP, i08u					32
			addc	Rc, Ra, Rb	W, cas, c0, c1		キャリー付き加算		16
			addhw	Rc, Ra, Rb	W, ovs		オーバーフローあり加算		16
			adds	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		32
			addsr	Rc, Ra, Rb			$Ra + Rb + 1 \rightarrow Rb$		32
			s1add	Rc, Ra, Rb Rc3, Ra3, Rb3			$Ra + Rb \rightarrow Rb$		16
			s2add	Rc, Ra, Rb Rc3, Ra3, Rb3			$Ra + Rb \rightarrow Rb$ (>2)		32
			addmak	Rc, Ra, Rb	R, BP0		$Ra + Rb \rightarrow Rb$		16
			addarvw	Rc, Ra, Rb					
			halfワード						
			fadddvh	Rc, Ra, Rb	W, ovs				
			vdcdh	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		
			vdcdhvh	Rc, Ra, Rb	W, ovs		$Ra + Rb \rightarrow Rb$		
			vsaddh	Rb, Ra, i08s			$Ra + Rb \rightarrow Rb$		
			vsaddsh	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		
			vsaddsrh	Rc, Ra, Rb			$Ra + Rb + 1 \rightarrow Rb$ (丸めあり)		
			vsaddhvc	Rc, Ra, Rb	R, VC				
			vsaddhvh	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		
			vsaddhvh	Rc, Ra, Rb	W, ovs		$Ra + Rb \rightarrow Rb$		
			vsaddh	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		
			vsaddhvh	Rc, Ra, Rb	W, ovs		$Ra + Rb \rightarrow Rb$		
			vsaddb	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		
			vsaddb	Rb, Ra, i08s			$Ra + Rb \rightarrow Rb$		
			vsaddsb	Rc, Ra, Rb			$Ra + Rb \rightarrow Rb$		
			vsaddsrh	Rc, Ra, Rb			$Ra + Rb + 1 \rightarrow Rb$ (丸めあり)		

【図 22】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
ALU sub 系	S I M D	ワード	sub	Rc,Rb,Ra Ra2,Ra2 Ra3,Rb3,Ra3					32 16
			rsub	Rb,Ra,i08s Ra2,Rb2 Ra2,i04s			即値 - Ra → Rb (Ra2)		32 16
			subc	Rc,Rb,Ra	W.cas.c0c1		キャリー付き		
			subvw	Rc,Rb,Ra	W.ovs		オーバーフローあり		
			subs	Rc,Rb,Ra			Ra - Rb → Rb 221		
			submsk	Rc,Rb,Ra	R.BP0		Rb CFRBP0 → Rb		
		ハーフワード	fsubvh	Rc,Rb,Ra					
		S I M D	vsubh	Rc,Rb,Ra			Ra 16 16 Rb 16 16 → 16 16 Rb	A	32
			vsubhvh	Rc,Rb,Ra	W.ovs		Ra 16 16 Rb 16 16 → 16 16 Rb		
			vsrsubh	Rb,Ra,i08s			即値 - 即値 16 16 → 16 16 Rb		
			vsubsh	Rc,Rb,Ra			Ra 16 16 Rb 16 16 → >>1 >>1 Rb		
			vxsubh	Rc,Rb,Ra			Ra 16 16 Rb 16 16 → 16 16 Rb		
			vxsubhvh	Rc,Rb,Ra	W.ovs		Ra 16 16 Rb 16 16 → 16 16 Rb		
			vhsuh	Rc,Rb,Ra			Ra 16 16 Rb 16 16 → 16 16 Rb		
			vhsuhvh	Rc,Rb,Ra	W.ovs		Ra 16 16 Rb 16 16 → 16 16 Rb		
			visuh	Rc,Rb,Ra			Ra 16 16 Rb 16 16 → 16 16 Rb		
			visuhvh	Rc,Rb,Ra	W.ovs		Ra 16 16 Rb 16 16 → 16 16 Rb		
		バイト	vsubb	Rc,Rb,Ra			(即値)		
			vsrsubb	Rb,Ra,i08s			Ra 8 8 8 8 Rb 8 8 8 8 → 8 8 8 8 Rb		
			vsusb	Rc,Rb,Ra	RVC				

【図 23】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 18
ALU logic 系	SING LE	ワード	and	Rc,Ra,Rb Rb,Ra,08u Ra2,Rb2			論理積	A	32
			andn	Rc,Ra,Rb Rb,Ra,08u Ra2,Rb2					18 32
			or	Rc,Ra,Rb Rb,Ra,08u Ra2,Rb2			論理和		18 32
			xor	Rc,Ra,Rb Rb,Ra,08u Ra2,Rb2			排他的論理和		18 32
ALU mov 系	SING LE	ワード	mov	Rb,Reg32 Reg32,Rb Rb2,Reg18 Reg18,Rb2 Ra2,Rb Ra2,18s Ra2,08s			Reg32 = TAR LR SVR PSR CFR MH0 MH1 ML0 ML1 EPSR IPC IPSR PC EPC PSR0 PSR1 PSR2 PSR3 CFR0 CFR1 CFR2 CFR3 Reg18 = TAR LR MH0 MH1	A	32
			movp	Rc,Rc+1,Ra,Rb			Rc ← Ra; Rc+1 ← Rb;		18 32
			movcf	Ck,Cj,Cm,Cn			Ck ← Cj; Cm ← Cn;		32
			mvcdovs	Cm,Cm+1	W:ovs		Cm,Cm+1 ← CFR.OVS; CFR.OVS; CFR.OVS ← 0;		18
			mvcdcas	Cm,Cm+1	W:cas		Cm,Cm+1 ← CFR.CAS; CFR.CAS; CFR.CAS ← 0;		32
			sethi	Ra,18s					32
ALU max min 系	SING LE	ワード	max	Rc,Ra,Rb	W:c0.c1		Rc ← max(Ra,Rb)	A	32
			min	Rc,Ra,Rb	W:c0.c1		Rc ← min(Ra,Rb)		32
		ハーフワード	vmaxh	Rc,Ra,Rb					32
			vminh	Rc,Ra,Rb					32
		バイト	vmaxb	Rc,Ra,Rb					32
			vminb	Rc,Ra,Rb					32
ALU abs 系	SING LE	ワード	abs	Rb,Ra			絶対値	A	32
			absevw	Rb,Ra	W:ovs		オーバーフローあり		32
			fabsvh	Rb,Ra	W:ovs				32
			vabshvh	Rb,Ra	W:ovs				32
ALU neg 系	SING LE	ワード	negvw	Rb,Ra	W:ovs			A	32
			fnegvh	Rb,Ra	W:ovs				32
			vmegvh	Rb,Ra	W:ovs				32
									32
ALU sum 系	SING LE	ワード	vsumh	Rb,Ra				A	32
			vsumh2	Rb,Ra					32
			vsumh2	Rb,Ra					32
			vabssumb	Rc,Ra,Rb					C
ALU その他	SING LE	ワード	fmdvh	Rb,Ra	W:ovs		丸め	C	32
			vfmdvh	Rb,Mn	W:ovs				32
			vsel	Rc,Ra,Rb	R:VC				32
			vsgnh	Rb,Ra					32

【図 24】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代数的な動作	演算器	31 16
CMP	S I N G L E		cmpCCn	Cm,Ra,Rb,Cn Cm,Ra,i05s,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, le, lt, leu, leu Cm <- result & Cn; (Cm+1 <- "result & Cn")	A	32
			cmpCCe	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result & Cn; Cm+1 <- "(result & Cn);"		
			cmpCCo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result Cn; Cm+1 <- "(result Cn);"		
			cmpCC	C6,Ra2,Rb2 C6,Ra2,i04s	W:CF		CC = eq, ne, gt, ge, le, lt C6 <- result		16
			tstzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; (Cm+1 <- "(Ra & Rb == 0) & Cn;")		32
			tstza	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; Cm+1 <- "(Ra & Rb == 0) & Cn;"		
			tstzo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) Cn; Cm+1 <- "(Ra & Rb == 0) Cn;"		
			tstzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; (Cm+1 <- "(Ra & Rb != 0) & Cn;")		
			tstna	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; Cm+1 <- "(Ra & Rb != 0) & Cn;"		16
			tstno	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) Cn; Cm+1 <- "(Ra & Rb != 0) Cn;"		
			tatz	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 <- (Ra2 & Rb2 == 0)		
			lstrn	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 <- (Ra2 & Rb2 != 0)		
S I M D	S I M D	ハーフワード	vcmpCCh vsmpCCh	Ra,Rb Ra,i05s	W:CF		CC = eq, ne, gt, le, ge, lt	A	32
		バイト	vcmpCCb vsmpCCb	Ra,Rb Ra,i05s	W:CF		CC = eq, ne, gt, le, ge, lt		

【図 25】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16	
mul系	SINGLE	ワード×ワード	mul	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s				X2	32	
			mulu	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s			符号なし乗算			
			fmulhww	Mm,Rc,Ra,Rb		fxp	固定小数点演算			
		ワード× ハーフワード	hmul	Mm,Rc,Ra,Rb				X1		
			lmul	Mm,Rc,Ra,Rb						
			fmulhww	Mm,Rc,Ra,Rb		fxp				
		ハーフワード× ハーフワード	fmulhw	Mm,Rc,Ra,Rb		fxp				
			fmulhh	Mm,Rc,Ra,Rb		fxp				
			fmulhhr	Mm,Rc,Ra,Rb		fxp	丸めあり			
		SIMD	ハーフワード × ハーフワード	vmul	Mm,Rc,Ra,Rb					X2
				vfmulw	Mm,Rc,Ra,Rb		fxp			
				vfmulh	Mm,Rc,Ra,Rb		fxp			
	vfmulhr			Mm,Rc,Ra,Rb		fxp	丸めあり			
	vumul	Mm,Rc,Ra,Rb								
	vxfmulw	Mm,Rc,Ra,Rb			fxp					
	vxfmulh	Mm,Rc,Ra,Rb			fxp					
vxfmulhr	Mm,Rc,Ra,Rb			fxp	丸めあり					
vhmul	Mm,Rc,Ra,Rb									
vhfmulw	Mm,Rc,Ra,Rb			fxp						
vhfmulh	Mm,Rc,Ra,Rb			fxp						
vhfmulhr	Mm,Rc,Ra,Rb			fxp	丸めあり					
vlmul	Mm,Rc,Ra,Rb									
vlfmulw	Mm,Rc,Ra,Rb			fxp						
vlfmulh	Mm,Rc,Ra,Rb			fxp						
vlfmulhr	Mm,Rc,Ra,Rb			fxp	丸めあり					
		ワード× ハーフワード	vpfmulhww	Mm,Rc,Rc+1,Ra,Rb Mm,Rc,Rc+1,Ra,Rb		fxp	丸めあり			

【図26】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 18	
mac系	SINGLE	ワード×ワード	mac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			mulを使った積和演算	X2	32	
			fmacww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulwwを使った積和演算			
		ワード × ハーフワード	hmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			hmulを使った積和演算	X1		
			lmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			lmulを使った積和演算			
		ハーフワード × ハーフワード	fmachww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhwwを使った積和演算			
			fmachw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhwを使った積和演算			
			fmachh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulhhを使った積和演算			
			fmachhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	丸めあり			
		SIMD	ハーフワード × ハーフワード	vmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmulを使った積和演算		X2
				vfmacw	Mm,Rc,Ra,Rb,Mn		fxp	vfmulhwを使った積和演算		
				vsmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vsmulを使った積和演算		
				vsmacw	Mm,Rc,Ra,Rb,Mn		fxp	vsmulhwを使った積和演算		
vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	vfmulhを使った積和演算				
vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	丸めあり				
vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx					vfmulを使った積和演算				
vfmachw	Mm,Rc,Ra,Rb,Mn				fxp	vfmulhwを使った積和演算				
vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	vfmulhを使った積和演算				
vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	丸めあり				
vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx					vfmulを使った積和演算				
vfmachw	Mm,Rc,Ra,Rb,Mn				fxp	vfmulhwを使った積和演算				
vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	vfmulhを使った積和演算				
vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	丸めあり				
vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	vfmulhを使った積和演算				
vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx				fxp	丸めあり				
ワード× ハーフワード	vpfmachww	Mm,Rc,Rc+1,Ra,Rb,Mn		fxp						

【図27】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 18
msu系	SINGLE	ワード×ワード	msu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			msuを使った積差演算	X2	32
			fmsuww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuwwを使った積差演算		
		ワード ×ハーフワード	hmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			hmsuを使った積差演算	X1	
			lmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			lmsuを使った積差演算		
		ハーフワード ×ハーフワード	fmsuww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuwwを使った積差演算		
			fmsuhw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuhwを使った積差演算		
		ハーフワード ×ハーフワード	fmsuhh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmsuhhを使った積差演算	丸めあり	
			fmsuhw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp			
		ハーフワード × ハーフワード	vmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmsuを使った積差演算	X2	
			vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vfmsuwを使った積差演算		
			vfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfmsuhを使った積差演算		
			vmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmsuを使った積差演算		
			vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vfmsuwを使った積差演算		
			vfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfmsuhを使った積差演算		
			vhmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vhmsuを使った積差演算		
			vhfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vhfmsuwを使った積差演算		
			vhfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vhfmsuhを使った積差演算		
			vfmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vfmsuを使った積差演算		
vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vfmsuwを使った積差演算					
vfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfmsuhを使った積差演算					

【図28】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
MEM Id 系	シ ン グ ル	ワード	ld	Rb.(Ra,d10u) Rb.(GP,d13u) Rb.(SP,d13u) Rb.(Rr+*10s) Rb2.(Ra2) Rb2.(Ra2,d05u) Rb2.(GP,d08u) Rb2.(SP,d08u) Rb2.(Ra2+)			レジスタ ← メモリ 32 ← 32	M	32
			ldh	Rb.(Ra,d09u) Rb.(GP,d12u) Rb.(SP,d12u) Rb.(Ra+*09s) Rb2.(Ra2) Rb3.(Ra3,d04u) Rb2.(GP,d05u) Rb2.(SP,d05u) Rb2.(Ra2+)			32 ← 16		16
			ldhu	Rb.(Ra,d09u) Rb.(GP,d12u) Rb.(SP,d12u) Rb.(Ra+*09s)					32
		バイト	ldb	Rb.(Ra,d08u) Rb.(GP,d11u) Rb.(SP,d11u) Rb.(Ra+*08s)			レジスタ ← メモリ 32 ← 8		16
			ldbu	Rb.(Ra,d08u) Rb.(GP,d11u) Rb.(SP,d11u) Rb.(Ra+*08s)					32
		バイト→ ハーフワード	ldbh	Rb.(Ra+*07s)			16 ← 8		32
			ldbuh	Rb.(Ra+*07s)					
		ワード	ldp	Rb:Rb+1.(Ra,d11u) LR:SVR.(Ra,d11u) TAR:UDR.(Ra,d14u) Rb:Rb+1.(GP,d14u) LR:SVR.(GP,d14u) TAR:UDR.(GP,d14u) Rb:Rb+1.(SP,d14u) LR:SVR.(SP,d14u) TAR:UDR.(SP,d14u) Rb:Rb+1.(Rr+*11s) Rb:Rb+1.(SP,d07u) LR:SVR.(SP,d07u) Rb2:Rr2.(Ra2+)			32 ← 32 32 ← 32		16
		ハーフ ワード	ldhp	Rb:Rb+1.(Ra,d10u) Rb:Rb+1.(Rr+*10s) Rb2:Rr2.(Ra2+)			32 ← 16		32
			ldbhp	Rb:Rb+1.(Ra,d09u) Rb:Rb+1.(Rr+*09s)			32 ← 8		16
		バイト→ ハーフ ワード	ldbhp	Rb:Rb+1.(Ra+*07s)			16 ← 8		32

【図 29】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	
MEM store 系	S I M D E	ワード	st	(Ra,d10u),Rb (GP,d13u),Rb (SP,d13u),Rb (Ra+),10s,Rb ----- (Ra2),Rb2 (Ra2,d05u),Rb2 (GP,d06u),Rb2 (SP,d06u),Rb2 (Ra2+),Rb2			レジスタ 32 → メモリ 32	M	32
							16		
							32		
							16		
		ハーフワード	sth	(Ra,d09u),Rb (GP,d12u),Rb (SP,d12u),Rb (Ra+),09s,Rb (Ra2),Rb2 (Ra2,d04u),Rb2 (GP,d05u),Rb2 (SP,d05u),Rb2 (Ra2+),Rb2			16 → 16		32
							16		
		バイト	stb	(Ra,d08u),Rb (GP,d11u),Rb (SP,d11u),Rb (Ra+),08s,Rb			8 → 8		
		バイト→ ハーフワード	stbh	(Ra+),07s,Rb			8 8 → 16		
		ワード ベ ア	stp	(Ra,d11u),Rb,Rb+1 (Ra,d11u),LR:SVR (Ra,d11u),TAR:UDR (GP,d14u),Rb,Rb+1 (GP,d14u),LR:SVR (GP,d14u),TAR:UDR (SP,d14u),Rb,Rb+1 (SP,d14u),LR:SVR (SP,d14u),TAR:UDR (Ra+),11s,Rb,Rb+1 ----- (SP,d07u),Rb,Rb (SP,d07u),LR:SVR (Ra2+),Rb2:Rb2			32 → 32 32 → 32		32
							16		
							16 16 → 32		32
							16		
							8 8 → 16		32
							8 8 8 8 → 32		
ハーフ ワード	sthp	(Ra,d10u),Rb,Rb+1 (Ra+),10s,Rb,Rb+1 ----- (Ra2+),Rb2:Rb2			16 16 → 32	32			
			16						
バイト	stbp	(Ra,d09u),Rb,Rb+1 (Ra+),09s,Rb,Rb+1			8 8 8 8 → 16	32			
バイト→ ハーフワード	stbhp	(Ra+),07s,Rb,Rb+1			8 8 8 8 → 32				

【図 30】

カテゴリ	SIMD サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算部	31 16
BRA		setlr	d09s C5.d09s			LRを設定 LRからフェッチした命令を分岐用バッファに格納	B	32 16
		settar	d09s C6.d09s C6.C2:C4.d09s C6.Cm.d09s C6.C4.d09s	W:c6 W:c2:c4:c6 W:c6:cm W:c6		TARを設定 TARからフェッチした命令を分岐用バッファに格納		32
		setbb	LR TAR			LRからフェッチした命令を分岐用バッファに格納 TARからフェッチした命令を分岐用バッファに格納		16
		jloop	C5.LR,Ra,i08s C6.TAR,Ra,i08s C6.C2:C4.TAR,Ra,i08s C6.Cm.TAR,Ra,i08s C6.TAR,Ra2 C6.C2:C4.TAR,Ra2 C6.Cm.TAR,Ra2	W:c5 W:c6 W:c2:c4:c6 W:c6:cm W:c6 W:c2:c4:c6 W:c6		プレディケート[c5]のみ プレディケート[c6]のみ		32
		jmp	TAR LR					16
		jmpf	TAR LR	R:CF				
		jmpf	TAR LR Cm.TAR C6.C2:C4.TAR					
		jmpf	LR					
		br	d20s d09s			プレディケート[c6][c7]のみ		32 16
		bri	d20s d09s	R:CF				32 16
		rti			W:PSR R:eh			

【図31】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BS asr系	SINGLE	ワード	asr	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			左シフト ←Ra(04) 	S1	32 16
			faslvr	Rc,Ra,Rb Rb,Ra,i05u Rc,Ra,Rb Rb,Ra,i05u	W.o.v.s		←Ra(04) 飽和あり 		
		ベアワード	aslp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			←Ra(04) 	S2	32
			faslpvr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u	W.o.v.s		←Ra(04) 飽和あり 		
	SIMD	ワード	vasl	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u				S2	32
			vfaslvr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u	W.o.v.s		←Ra(04) 飽和あり 		
		ハーフワード	vaslh	Rc,Ra,Rb Rb,Ra,i04u			←Ra(03) ←Ra(03) 	S1	32
			vfaslhv	Rc,Ra,Rb Rb,Ra,i04u	W.o.v.s		←Ra(03) ←Ra(03) 飽和あり 		
		バイト	vaslb	Rc,Ra,Rb Rb,Ra,i03u			←Ra(02) ←Ra(02) 		
							←Ra(02) ←Ra(02) 		
BS asr系	SINGLE	ワード	asr	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			算術右シフト >>Ra(04) 	S1	32 16
		ベアワード	asrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>Ra(04) 	S2	32
	SIMD	ワード	vasr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u				S1	32
		ハーフワード	vasrh	Rc,Ra,Rb Rb,Ra,i04u			>>Ra(03) >>Ra(03) 		
		バイト	vasrb	Rc,Ra,Rb Rb,Ra,i03u			>>Ra(02) >>Ra(02) 		
							>>Ra(02) >>Ra(02) 		

【図 34】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
CNV	SINGLE		bcnt1	Rb, Ra			1の数をカウント	C	32
			bseq0	Rb, Ra			MSBから最初に0になるまでをカウント		
			bseq1	Rb, Ra			MSBから最初に1になるまでをカウント		
			bseq	Rb, Ra			MSB-1から最初にMSBになるまでをカウント		
			msktbrvh	Rc, Ra, Rb	RBP0				
			byterev	Rb, Ra					
			msktbrvb	Rc, Ra, Rb	RBP0				
	SIMD	ハーフワード	vintllh	Rc, Ra, Rb					
			vintlhh	Rc, Ra, Rb					
		バイト	vintllb	Rc, Ra, Rb					
			vintlhb	Rc, Ra, Rb					
		ハーフワード	vhunpkh	Rb:Rb+1, Ra					
			vhunpkb	Rb:Rb+1, Ra					
		ハーフワード	vlunpkh	Rb:Rb+1, Ra					
			vlunpkhu	Rb:Rb+1, Ra					
		バイト	vlunpkb	Rb:Rb+1, Ra					
			vlunpkbu	Rb:Rb+1, Ra					
		ハーフワード	vunpk1	Rb, Mn					
			vunpk2	Rb, Mn					
			vstovh	Rb, Ra					
		バイト	vstovb	Rb, Ra					
			vhpkb	Rc, Ra, Rb					

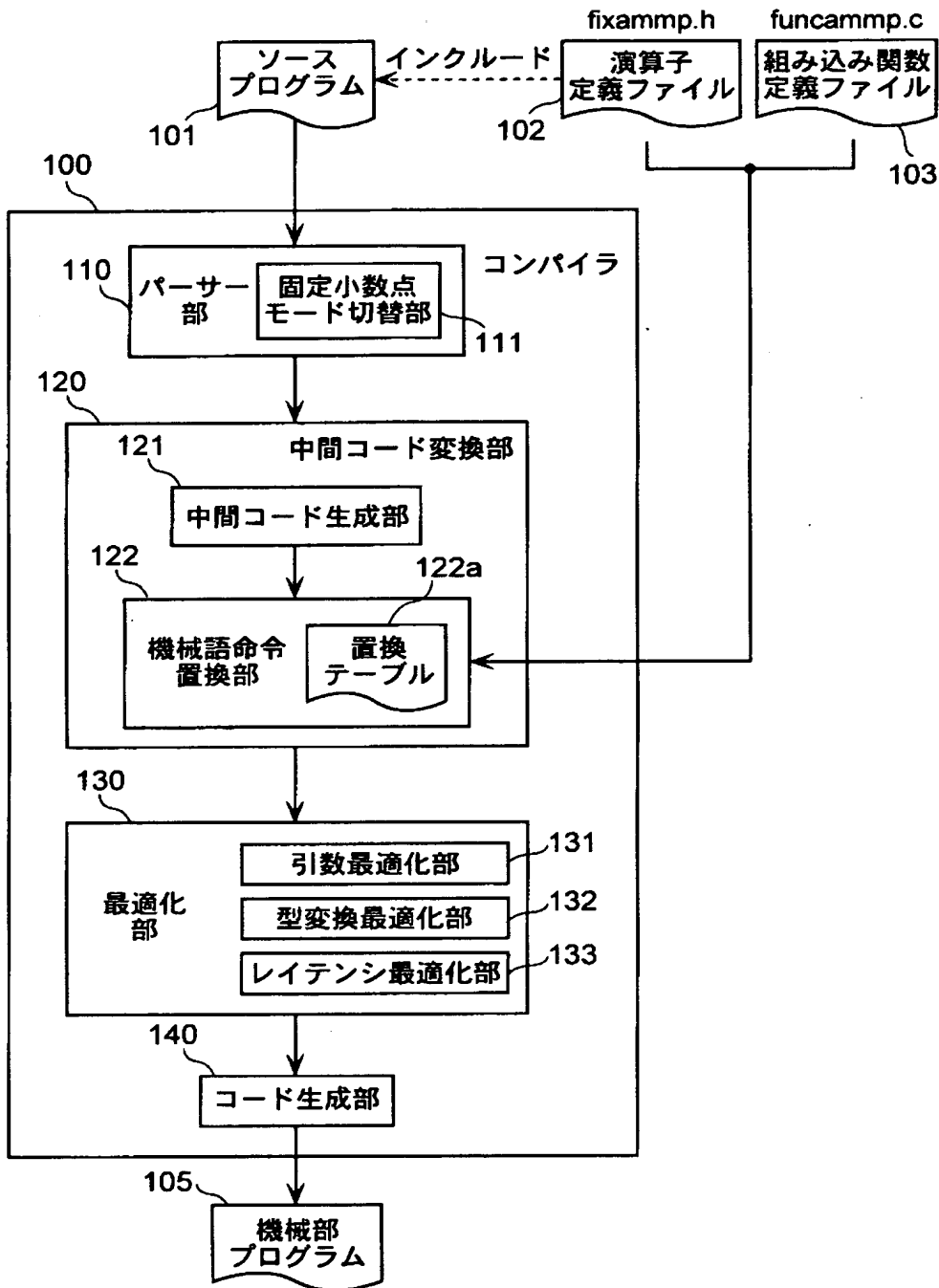
【図 35】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
SAT vphk系	SIMD	ワード→ ハーフワード	vphkh	Rc,Ra,Rb				C	32
			vphkuh	Rc,Ra,Rb					
		ハーフワード→ バイト	vphkb	Rc,Ra,Rb					
			vphkbu	Rc,Ra,Rb					
SAT sat系	SIMD	ワード	satw	Mm,Rb,Mn			ワード飽和	C	32
			sath	Rb,Ra			ハーフワード飽和		
			satb	Rb,Ra			バイト飽和		
			satbu	Rb,Ra			バイト符号なし飽和		
			sat9	Rb,Ra			9ビット飽和		
			sat12	Rb,Ra			12ビット飽和		
	SIMD	ハーフワード	vsath	Mm,Rb,Mn					
			vsath8	Rb,Ra			符号付き8ビット飽和		
			vsath8u	Rb,Ra			符号なし8ビット飽和		
			vsath9	Rb,Ra			9ビット飽和		
			vsath12	Rb,Ra			12ビット飽和		

【図 36】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
MSK			mskgen	Rc,Rb Rb,JO5U,JO5u			マスク生成 	S2	32
			msk	Rc,Ra,Rb Rb,Ra,JO5U,JO5u					
EXTR			extr	Rc,Ra,Rb Rb,Ra,JO5U,JO5u			符号拡張あり 	S2	32
			extru	Rc,Ra,Rb Rb,Ra,JO5U,JO5u			符号拡張なし 		
DIV			div divu	MHm,Rc,MHn,Ra,Rb MHm,Rc,MHn,Ra,Rb	W:ovs		除算	DIV	32
ETC			piNl			W:hi,ie,pl R:PSR	ソフトウェア割り込みN=0~7	B	32
			piN			W:hi,ie,pl R:PSR	ソフトウェア割り込みN=0~7		16
			scN			W:hi,ie,pl R:PSR	システムコールN=0~7		
			ldstb	Rb,(Ra)			loadバスロック	M	32
			rd	Rb,(Ra) Rb,(d11u) Rb2,(Ra2)		R:see	外部レジスタリード		16
			wt	(Ra),Rb (d11u),Rb (Ra2),Rb2		R:see	外部レジスタライト		32
			dpref	(Ra,d11u)			プリフェッチ	B	16
			dbgmn	i18u			N=0~3		DBGM
			vcch		W:CF R:VC		VCフラグチェック		
			vmpsw	LR			VMP切替え	B	32
			vmpsw	LR			VMP切替え禁止		
			vmpintd1			W:ie	VMP切替え許可		
			vmpintd2			W:ie	VMP切替え許可		
			vmpintd3			W:ie	VMP切替え許可		
			vmpinte1				VMP切替え許可	A	16
			vmpinte2				VMP切替え許可		
			vmpinte3				VMP切替え許可	A	16
			nop				no operation		

【図 37】



【図 3 8】

```

/*****
*
* (C)Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*   fixamp.h
*   Version:
*   Release:
*   Date:      2002/6/14   v0.9.1 対応
*
*****/

/* 多重定義回避 */
#ifndef __FIXAMP__
#define __FIXAMP__

/* FIX-Lib. クラス定義 */
class FIX16_1;
class FIX32_1;
class FIX16_2;
class FIX32_2;

#ifdef __AMPPCC__
#pragma pack_struct
#endif // __AMPPCC__

////////////////////////////////////
//                               Member of FIX16_1
//                               val : 実際の 16bit の値
////////////////////////////////////
class FIX16_1{
    short val;
public:
    // constructor
    FIX16_1(){}
    FIX16_1(int a);
    FIX16_1(float a);
    FIX16_1(double a);
    FIX16_1(FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(volatile FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(const FIX16_1& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_1& operator=(FIX16_1 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_1& operator=(FIX16_1 a)
    {

```

【図 3 9】

```

        val = a.val;
        return *this;
    }
    friend FIX16_1 operator+(FIX16_1 a);
    friend FIX16_1 operator-(FIX16_1 a);

    friend FIX16_1 operator+(FIX16_1 a, FIX16_1 b);
    friend FIX16_1 operator-(FIX16_1 a, FIX16_1 b);

    friend FIX16_1 operator*(FIX16_1 a, FIX16_1 b);
    friend FIX16_1 operator*(int a, FIX16_1 b);
    friend FIX16_1 operator*(FIX16_1 a, int b);
    friend FIX16_1 operator*(float a, FIX16_1 b);
    friend FIX16_1 operator*(FIX16_1 a, float b);
    friend FIX16_1 operator*(double a, FIX16_1 b);
    friend FIX16_1 operator*(FIX16_1 a, double b);

    friend FIX16_1 operator/(FIX16_1 a, FIX16_1 b);
    friend FIX16_1 operator/(FIX16_1 a, int b);
    friend FIX16_1 operator/(FIX16_1 a, float b);
    friend FIX16_1 operator/(FIX16_1 a, double b);

    friend FIX16_1 operator<<(FIX16_1 a, int b);
    friend FIX16_1 operator>>(FIX16_1 a, int b);

    friend bool operator<(FIX16_1 a, FIX16_1 b);
    friend bool operator>(FIX16_1 a, FIX16_1 b);
    friend bool operator<=(FIX16_1 a, FIX16_1 b);
    friend bool operator>=(FIX16_1 a, FIX16_1 b);
    friend bool operator==(FIX16_1 a, FIX16_1 b);
    friend bool operator!=(FIX16_1 a, FIX16_1 b);

    volatile FIX16_1& operator<<=(int b) volatile;
        FIX16_1& operator<<=(int b);
    volatile FIX16_1& operator>>=(int b) volatile;
        FIX16_1& operator>>=(int b);

    volatile FIX16_1& operator*=(FIX16_1 b) volatile;
        FIX16_1& operator*=(FIX16_1 b);
    volatile FIX16_1& operator*=(int b) volatile;
        FIX16_1& operator*=(int b);
    volatile FIX16_1& operator*=(float b) volatile;
        FIX16_1& operator*=(float b);
    volatile FIX16_1& operator*=(double b) volatile;
        FIX16_1& operator*=(double b);

    volatile FIX16_1& operator/=(FIX16_1 b) volatile;
        FIX16_1& operator/=(FIX16_1 b);
    volatile FIX16_1& operator/=(int b) volatile;
        FIX16_1& operator/=(int b);
    volatile FIX16_1& operator/=(float b) volatile;
        FIX16_1& operator/=(float b);
    volatile FIX16_1& operator/=(double b) volatile;
        FIX16_1& operator/=(double b);
    volatile FIX16_1& operator+=(FIX16_1 b) volatile;
        FIX16_1& operator+=(FIX16_1 b);
    volatile FIX16_1& operator-=(FIX16_1 b) volatile;
        FIX16_1& operator-=(FIX16_1 b);

```

【図 40】

```

short value(){return val;}

// Other functions

friend FIX16_1 _fix161(short a);
friend short _bptn(FIX16_1 a);
friend FIX16_1 _fix161(FIX32_1 a);
friend float _fToat(FIX16_1 a);
friend double _double(FIX16_1 a);

friend FIX16_1 _abs(FIX16_1 a);
friend FIX16_1 _max(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _min(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _adds(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _subs(FIX16_1 a, FIX16_1 b);
friend int _bcnt1(FIX16_1 a);
friend int _bseq(FIX16_1 a);
friend int _bseq0(FIX16_1 a);
friend int _bseq1(FIX16_1 a);
friend FIX16_1 _round(FIX32_1 a);
friend int _extr(FIX16_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_1 a, unsigned int b, unsigned int c);

friend void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mulr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _macr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _msur(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, int b);
friend FIX16_1 _div(FIX16_1 a, float b);
friend FIX16_1 _div(FIX16_1 a, double b);

};

////////////////////////////////////
//                               Member of FIX32_1
//
//                               val : 実際の 32bit の値
//
////////////////////////////////////
class FIX32_1{
    long val;
public:
    // constructor
    FIX32_1(){};
    FIX32_1(int a);

```

【図 4 1】

```

FIX32_1(float a);
FIX32_1(double a);
FIX32_1(FIX16_1 a);
FIX32_1(FIX32_1& a)
{
    val = a.val;
}
FIX32_1(volatile FIX32_1& a)
{
    val = a.val;
}
FIX32_1(const FIX32_1& a)
{
    val = a.val;
}

// Operator
volatile FIX32_1& operator=(FIX32_1 a) volatile
{
    val = a.val;
    return *this;
}
FIX32_1& operator=(FIX32_1 a)
{
    val = a.val;
    return *this;
}

friend FIX32_1 operator+(FIX32_1 a);
friend FIX32_1 operator-(FIX32_1 a);

friend FIX32_1 operator+(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator-(FIX32_1 a, FIX32_1 b);

friend FIX32_1 operator*(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator*(int a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, int b);
friend FIX32_1 operator*(float a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, float b);
friend FIX32_1 operator*(double a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, double b);

friend FIX32_1 operator/(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator/(FIX32_1 a, int b);
friend FIX32_1 operator/(FIX32_1 a, float b);
friend FIX32_1 operator/(FIX32_1 a, double b);

friend FIX32_1 operator<<(FIX32_1 a, int b);
friend FIX32_1 operator>>(FIX32_1 a, int b);

friend bool operator<(FIX32_1 a, FIX32_1 b);
friend bool operator>(FIX32_1 a, FIX32_1 b);
friend bool operator<=(FIX32_1 a, FIX32_1 b);
friend bool operator>=(FIX32_1 a, FIX32_1 b);
friend bool operator==(FIX32_1 a, FIX32_1 b);
friend bool operator!=(FIX32_1 a, FIX32_1 b);

volatile FIX32_1& operator<=(int b) volatile;
FIX32_1& operator<=(int b);
volatile FIX32_1& operator>=(int b) volatile;

```

【図 4 2】

```

        FIX32_1& operator>>=(int b);

volatile FIX32_1& operator*=(FIX32_1 b) volatile;
        FIX32_1& operator*=(FIX32_1 b);
volatile FIX32_1& operator*=(int b) volatile;
        FIX32_1& operator*=(int b);
volatile FIX32_1& operator*=(float b) volatile;
        FIX32_1& operator*=(float b);
volatile FIX32_1& operator*=(double b) volatile;
        FIX32_1& operator*=(double b);

volatile FIX32_1& operator/=(FIX32_1 b) volatile;
        FIX32_1& operator/=(FIX32_1 b);
volatile FIX32_1& operator/=(int b) volatile;
        FIX32_1& operator/=(int b);
volatile FIX32_1& operator/=(float b) volatile;
        FIX32_1& operator/=(float b);
volatile FIX32_1& operator/=(double b) volatile;
        FIX32_1& operator/=(double b);

volatile FIX32_1& operator+=(FIX32_1 b) volatile;
        FIX32_1& operator+=(FIX32_1 b);
volatile FIX32_1& operator-=(FIX32_1 b) volatile;
        FIX32_1& operator-=(FIX32_1 b);

long value(){return val;}

// Other functions
friend FIX32_1 _fix321(long a);
friend long _bptn(FIX32_1 a);
friend float _float(FIX32_1 a);
friend double _double(FIX32_1 a);

friend FIX32_1 _abs(FIX32_1 a);
friend FIX32_1 _max(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _min(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _adds(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _subs(FIX32_1 a, FIX32_1 b);
friend int _bcnt1(FIX32_1 a);
friend int _bseq(FIX32_1 a);
friend int _bseq0(FIX32_1 a);
friend int _bseq1(FIX32_1 a);
friend FIX16_1 _round(FIX32_1 a);

friend int _extr(FIX32_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_1 a, unsigned int b, unsigned int c);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);

friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

```

【図 4 3】

```

friend void    _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, int b);
friend FIX32_1 _div(FIX32_1 a, float b);
friend FIX32_1 _div(FIX32_1 a, double b);

};

////////////////////////////////////
//                               Member of FIX16_2
//
//                               val : 実際の 16bit の値
//
////////////////////////////////////
class FIX16_2{
    short val;
public:
    // constructor
    FIX16_2(){}
    FIX16_2(int a);
    FIX16_2(float a);
    FIX16_2(double a);
    FIX16_2(FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(volatile FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(const FIX16_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_2& operator=(FIX16_2 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_2& operator=(FIX16_2 a)
    {
        val = a.val;
        return *this;
    }
    friend FIX16_2 operator+(FIX16_2 a);
    friend FIX16_2 operator-(FIX16_2 a);

    friend FIX16_2 operator+(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator-(FIX16_2 a, FIX16_2 b);

    friend FIX16_2 operator*(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator*(int    a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, int    b);
    friend FIX16_2 operator*(float  a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, float  b);
    friend FIX16_2 operator*(double a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, double b);

```

【図 4 4】

```

friend FIX16_2 operator/(FIX16_2 a, FIX16_2 b);
friend FIX16_2 operator/(FIX16_2 a, int b);
friend FIX16_2 operator/(FIX16_2 a, float b);
friend FIX16_2 operator/(FIX16_2 a, double b);

friend FIX16_2 operator<<(FIX16_2 a, int b);
friend FIX16_2 operator>>(FIX16_2 a, int b);

friend bool operator<(FIX16_2 a, FIX16_2 b);
friend bool operator>(FIX16_2 a, FIX16_2 b);
friend bool operator<=(FIX16_2 a, FIX16_2 b);
friend bool operator>=(FIX16_2 a, FIX16_2 b);
friend bool operator==(FIX16_2 a, FIX16_2 b);
friend bool operator!=(FIX16_2 a, FIX16_2 b);

volatile FIX16_2& operator<=(int b) volatile;
FIX16_2& operator<=(int b);
volatile FIX16_2& operator>=(int b) volatile;
FIX16_2& operator>=(int b);

volatile FIX16_2& operator*=(FIX16_2 b) volatile;
FIX16_2& operator*=(FIX16_2 b);
volatile FIX16_2& operator*=(int b) volatile;
FIX16_2& operator*=(int b);
volatile FIX16_2& operator*=(float b) volatile;
FIX16_2& operator*=(float b);
volatile FIX16_2& operator*=(double b) volatile;
FIX16_2& operator*=(double b);

volatile FIX16_2& operator/=(FIX16_2 b) volatile;
FIX16_2& operator/=(FIX16_2 b);
volatile FIX16_2& operator/=(int b) volatile;
FIX16_2& operator/=(int b);
volatile FIX16_2& operator/=(float b) volatile;
FIX16_2& operator/=(float b);
volatile FIX16_2& operator/=(double b) volatile;
FIX16_2& operator/=(double b);

volatile FIX16_2& operator+=(FIX16_2 b) volatile;
FIX16_2& operator+=(FIX16_2 b);
volatile FIX16_2& operator-=(FIX16_2 b) volatile;
FIX16_2& operator-=(FIX16_2 b);

short value(){return val;}

// Other functions
friend FIX16_2 _fix162(short a);
friend short _bptn(FIX16_2 a);
friend FIX16_2 _fix162(FIX32_2 a);
friend float _float(FIX16_2 a);
friend double _double(FIX16_2 a);

friend FIX16_2 _abs(FIX16_2 a);
friend FIX16_2 _max(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _min(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _adds(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _subs(FIX16_2 a, FIX16_2 b);

```


【図 4 5】

```

friend int      _bcnt1(FIX16_2 a);
friend int      _bseq(FIX16_2 a);
friend int      _bseq0(FIX16_2 a);
friend int      _bseq1(FIX16_2 a);
friend FIX16_2  _round(FIX32_2 a);

friend int _extr(FIX16_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_2 a, unsigned int b, unsigned int c);

friend void      _mul(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void      _msu(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend FIX16_2   _div(FIX16_2 a, FIX16_2 b);
friend FIX16_2   _div(FIX16_2 a, int b);
friend FIX16_2   _div(FIX16_2 a, float b);
friend FIX16_2   _div(FIX16_2 a, double b);

};

////////////////////////////////////
//                               Member of FIX32_2
//
//                               val : 実際の 32bit の値
//
////////////////////////////////////
class FIX32_2{
    long val;
public:
    // constructor
    FIX32_2(){}
    FIX32_2(int a);
    FIX32_2(float a);
    FIX32_2(double a);
    FIX32_2(FIX16_2 a);
    FIX32_2(FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(volatile FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(const FIX32_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX32_2& operator=(FIX32_2 a) volatile
    {

```

【図 4 6】

```

    val = a.val;
    return *this;
}
FIX32_2& operator=(FIX32_2 a)
{
    val = a.val;
    return *this;
}
friend FIX32_2 operator+(FIX32_2 a);
friend FIX32_2 operator-(FIX32_2 a);

friend FIX32_2 operator+(FIX32_2 a, FIX32_2 b);
friend FIX32_2 operator-(FIX32_2 a, FIX32_2 b);

friend FIX32_2 operator*(FIX32_2 a, FIX32_2 b);
friend FIX32_2 operator*(int a, FIX32_2 b);
friend FIX32_2 operator*(FIX32_2 a, int b);
friend FIX32_2 operator*(float a, FIX32_2 b);
friend FIX32_2 operator*(FIX32_2 a, float b);
friend FIX32_2 operator*(double a, FIX32_2 b);
friend FIX32_2 operator*(FIX32_2 a, double b);

friend FIX32_2 operator/(FIX32_2 a, FIX32_2 b);
friend FIX32_2 operator/(FIX32_2 a, int b);
friend FIX32_2 operator/(FIX32_2 a, float b);
friend FIX32_2 operator/(FIX32_2 a, double b);

friend FIX32_2 operator<<(FIX32_2 a, int b);
friend FIX32_2 operator>>(FIX32_2 a, int b);

friend bool operator<(FIX32_2 a, FIX32_2 b);
friend bool operator>(FIX32_2 a, FIX32_2 b);
friend bool operator<=(FIX32_2 a, FIX32_2 b);
friend bool operator>=(FIX32_2 a, FIX32_2 b);
friend bool operator==(FIX32_2 a, FIX32_2 b);
friend bool operator!=(FIX32_2 a, FIX32_2 b);

volatile FIX32_2& operator<=(int b) volatile;
FIX32_2& operator<=(int b);
volatile FIX32_2& operator>=(int b) volatile;
FIX32_2& operator>=(int b);

volatile FIX32_2& operator*=(FIX32_2 b) volatile;
FIX32_2& operator*=(FIX32_2 b);
volatile FIX32_2& operator*=(int b) volatile;
FIX32_2& operator*=(int b);
volatile FIX32_2& operator*=(float b) volatile;
FIX32_2& operator*=(float b);
volatile FIX32_2& operator*=(double b) volatile;
FIX32_2& operator*=(double b);

volatile FIX32_2& operator/=(FIX32_2 b) volatile;
FIX32_2& operator/=(FIX32_2 b);
volatile FIX32_2& operator/=(int b) volatile;
FIX32_2& operator/=(int b);
volatile FIX32_2& operator/=(float b) volatile;
FIX32_2& operator/=(float b);
volatile FIX32_2& operator/=(double b) volatile;
FIX32_2& operator/=(double b);

```

【図 4 7】

```

volatile FIX32_2& operator+=(FIX32_2 b) volatile;
FIX32_2& operator+=(FIX32_2 b);
volatile FIX32_2& operator-=(FIX32_2 b) volatile;
FIX32_2& operator-=(FIX32_2 b);

long value(){return val;}

// Other functions
friend FIX32_2 _fix322(long a);
friend long _bptn(FIX32_2 a);
friend float _float(FIX32_2 a);
friend double _double(FIX32_2 a);

friend FIX32_2 _abs(FIX32_2 a);
friend FIX32_2 _max(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _min(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _adds(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _subs(FIX32_2 a, FIX32_2 b);
friend int _bcnt1(FIX32_2 a);
friend int _bseq(FIX32_2 a);
friend int _bseq0(FIX32_2 a);
friend int _bseq1(FIX32_2 a);
friend FIX16_2 _round(FIX32_2 a);

friend int _extr(FIX32_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_2 a, unsigned int b, unsigned int c);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);

friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, int b);
friend FIX32_2 _div(FIX32_2 a, float b);
friend FIX32_2 _div(FIX32_2 a, double b);

};

#if defined(__AMPPCC__)
#pragma pack_struct_default
#endif // __AMPPCC__

// other functions
#if defined(__AMPPCC__)
#pragma _enable_asm_begin
static inline FIX16_1 _fix161(FIX32_1 a)
{
    FIX16_1 result;

```

【図 4 8】

```
asm(vr0 = a){
    asr vr1, vr0, 16;
}(result = vr1);

return result;
}

static inline FIX16_2 _fix162(FIX32_2 a)
{
    FIX16_2 result;

    asm(vr0 = a){
        asr vr1, vr0, 16;
    }(result = vr1);

    return result;
}

static inline FIX16_1 _fix161(short a)
{
    FIX16_1 result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_2 _fix162(short a)
{
    FIX16_2 result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX32_1 _fix321(long a)
{
    FIX32_1 result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX32_2 _fix322(long a)
{
    FIX32_2 result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);
```

【図 4 9】

```
    return result;
}

static inline short _bptn(FIX16_1 a)
{
    short result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline short _bptn(FIX16_2 a)
{
    short result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline long _bptn(FIX32_1 a)
{
    long result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline long _bptn(FIX32_2 a)
{
    long result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_1 _abs(FIX16_1 a)
{
    FIX16_1 result;

    asm(vr0 = a){
        absvh vr1, vr0;
    }(result = vr1);

    return result;
}
```

【図 5 0】

```
static inline FIX32_1 _abs(FIX32_1 a)
{
    FIX32_1 result;

    asm(vr0 = a){
        absvw vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_2 _abs(FIX16_2 a)
{
    FIX16_2 result;

    asm(vr0 = a){
        absvh vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX32_2 _abs(FIX32_2 a)
{
    FIX32_2 result;

    asm(vr0 = a){
        absvw vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_1 _max(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b){
        max vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX32_1 _max(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b){
        max vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX16_2 _max(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;
```

【図 5 1】

```
asm(vr0 = a, vr1 = b){
    max vr2, vr0, vr1;
}(result = vr2);

    return result;
}

static inline FIX32_2 _max(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b){
        max vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX16_1 _min(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b){
        min vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX32_1 _min(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b){
        min vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX16_2 _min(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b){
        min vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX32_2 _min(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b){
        min vr2, vr0, vr1;
```

【図 5 2】

```
    }(result = vr2);  
    return result;  
}  
  
static inline FIX16_1 _adds(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b){  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_1 _adds(FIX32_1 a, FIX32_1 b)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a, vr1 = b){  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_2 _adds(FIX16_2 a, FIX16_2 b)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a, vr1 = b){  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _adds(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b){  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_1 _subs(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b){  
        subs    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}
```


【図 5 3】

```
static inline FIX32_1 _subs(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX16_2 _subs(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX32_2 _subs(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline int _bcnt1(FIX16_1 a)
{
    int result;

    asm(vr0 = a){
        bcnt1    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bcnt1(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bcnt1    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bcnt1(FIX32_1 a)
{
    int result;
```

【図 5 4】

```
asm(vr0 = a){
    bcnt1    vr1, vr0;
}(result = vr1);

return result;
}

static inline int _bcnt1(FIX32_2 a)
{
    int result;

    asm(vr0 = a){
        bcnt1    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq(FIX16_1 a)
{
    int result;

    asm(vr0 = a){
        bseq     vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq0(FIX16_1 a)
{
    int result;

    asm(vr0 = a){
        bseq0    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX16_1 a)
{
    int result;

    asm(vr0 = a){
        bseq1    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq     vr1, vr0;
    }(result = vr1);
```

【図 5 5】

```
    } return result;

static inline int _bseq0(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq0 vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq1 vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq0(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq0 vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq1 vr1, vr0;
    }(result = vr1);

    return result;
}
```

【図 5 6】

```

static inline int _bseq(FIX32_2 a)
{
    int result;

    asm(vr0 = a){
        bseq    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq0(FIX32_2 a)
{
    int result;

    asm(vr0 = a){
        bseq0   vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX32_2 a)
{
    int result;

    asm(vr0 = a){
        bseq1   vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_1 _round(FIX32_1 a)
{
    FIX16_1 result;

    asm(vr0 = a){
        rndvh   vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_2 _round(FIX32_2 a)
{
    FIX16_2 result;

    asm(vr0 = a){
        rndvh   vr1, vr0;
    }(result = vr1);

    return result;
}

static inline void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0, vr2, vr0, vr1;
    }
}

```

【図 5 7】

```

    } (c = vr2);
}

static inline void _mulr(long &mh, long&m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr1,vr0;
    } (mh = mh0, m1 = m10, c = vr2);
}

```

【図 5 8】

```

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr1,vr0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {

```

【図 5 9】

```

    fmachww m0, vr2, vr0, vr1, m0;
  } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _macr(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachhr m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

【図 6 0】

```

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0,vr2,vr1,vr0,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0,vr2,vr1,vr0,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmsuw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmsuw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msur(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhr m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

inline FIX16_1 operator/(FIX16_1 a, FIX16_1 b)
{

```


【図 6 1】

```

FIX16_1 result;

asm(vr0 = a, vr1 = b) {
    extw m0, vr3, vr0;
    aslp m0, vr4, mh0, vr3, 15;
    div  mh1, vr5, mh0, vr4, vr1;
    sath vr2, vr5;
} (result = vr2);

return result;
}

inline FIX16_2 operator/(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        aslp m0, vr4, mh0, vr3, 14;
        div  mh1, vr5, mh0, vr4, vr1;
        sath vr2, vr5;
    } (result = vr2);

    return result;
}

inline FIX32_1 operator/(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        mskgen vr3, 31, 31; //vr3 = 0x80000000
        cmpeq  C0:C1, vr1, vr3;
        [C0]   negvw vr2, vr0; //vr1==(-1)の場合、符号反転
        [C2]   cmpgtn C2:C3, vr0, 0, C1; //vr1!=(-1)の場合
        [C3]   negvw vr4, vr0; //vr0を負に符号反転(vr0')
        [C3]   mov vr4, vr0;
        [C2]   cmpgtn C2:C3, vr1, 0, C1;
        [C3]   negvw vr5, vr1; //vr1を負に符号反転(vr1')
        [C3]   mov vr5, vr1;
        [C2]   cmplen C2:C3, vr4, vr5, C1;
        [C2]   lsr vr6, vr0, 31; //vr0' <= 2vr1' (ovf 判定)の場合オーバーフロ
        [C2]   lsr vr7, vr1, 31;
        [C2]   xor vr8, vr6, vr7;
        [C4]   cmpeqn C4:C5, vr8, 0, C2;
        [C5]   mskgen vr2, 30, 0; //vr2 = 0x7fffffff
        [C5]   mskgen vr2, 31, 31; //vr2 = 0x80000000
        [C3]   extw m0, vr9, vr0; //被除数<除数の場合
        [C3]   aslp m0, vr10, mh0, vr9, 31;
        [C3]   div mh1, vr2, mh0, vr10, vr1;
    } (result = vr2);

    return result;
}

```

【図 6 2】

```

inline FIX32_2 operator/(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;
    asm(vr0 = a, vr1 = b) {
        mskgen    vr3,31,31; //vr3 = 0x80000000
        mskgen    vr4,31,30; //vr4 = 0xc0000000
        cmpgt     C0:C1,vr0,0;
        [C0]      negvw    vr5,vr0; //vr0 を負に符号反転(vr0')
        [C1]      mov      vr5,vr0;
        cmpgt     C0:C1,vr1,0;
        [C0]      negvw    vr6,vr1; //vr1 を負に符号反転(vr1')
        [C1]      mov      vr6,vr1;
        cmpeq     C0:C1,vr0,vr3;
        cmplt     C0:C1,vr6,vr4,C0; //vr0=-2 かつ vr1' < 0xc0000000 か?
        [C1]      asl     vr6,vr6,1; //!(vr0=-2 かつ vr1' < 0xc0000000)のとき
        cmple     C2:C3,vr5,vr6,C1;
        [C2]      lsr     vr7,vr0,31; //!(vr0=-2 かつ vr1' < 0xc0000000) かつ
        [C2]      lsr     vr8,vr1,31; // vr0' <= 2vr1' (ovf 判定) の場合
        [C2]      xor     vr9,vr7,vr8; //オーバーフロー
        cmpeq     C4:C5,vr9,0,C2;
        [C4]      mskgen    vr2,30,0; //vr2 = 0x7fffffff
        [C5]      mskgen    vr2,31,31; //vr2 = 0x80000000
        [C3]      extw     m0,vr10,vr0; //それ以外
        [C3]      asl     m0,vr11,mh0,vr10,30;
        [C3]      div     mh1,vr2,mh0,vr11,vr1;
    } (result = vr2);
    return result;
}

inline FIX16_1 operator/(FIX16_1 a, int b)
{
    FIX16_1 result;
    asm(vr0 = a, vr1 = b) {
        extw m0,vr3,vr0;
        div mh1,vr4,mh0,vr3,vr1;
        sath vr2,vr4;
    } (result = vr2);
    return result;
}

inline FIX16_2 operator/(FIX16_2 a, int b)
{
    FIX16_2 result;
    asm(vr0 = a, vr1 = b) {
        extw m0,vr3,vr0;
        div mh1,vr4,mh0,vr3,vr1;
        sath vr2,vr4;
    } (result = vr2);
    return result;
}

```

【図 6 3】

```

inline FIX32_1 operator/(FIX32_1 a, int b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        [C0] cmpeq    C0:C1, vr1, 1;
        [C1] mov      vr2, vr0;    //除数=1の時、Rc = Ra
        [C1] mskgen   vr3, 31, 31; //0x80000000
        [C1] cmpeqn   C2:C3, vr0, vr3, C1;
        [C4] cmpeqa   C4:C5, vr1, -1, C2;
        [C4] mskgen   vr2, 30, 0;  //被除数=-1 かつ 除数=-1 の時、Rc =
0x7fffffff
        [C5] extw     m0, vr4, vr0; //それ以外
        [C5] div      mh1, vr2, mh0, vr4, vr1;
    } (result = vr2);

    return result;
}

inline FIX32_2 operator/(FIX32_2 a, int b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen   vr3, 31, 31; //Rd = 0x80000000
        mskgen   vr4, 31, 30; //Re = 0xc0000000
        [C0] cmpgt   C0:C1, vr0, 0;
        [C1] negvw    vr5, vr0;    //Ra を負に符号反転(Ra')
        [C1] mov      vr5, vr0;
        [C1] aslww    vr6, vr1, 30; //int-->FIX32_2
        [C0] cmpgt    C0:C1, vr6, 0;
        [C1] negvw    vr7, vr6;    //Rb を負に符号反転(Rb')
        [C1] mov      vr7, vr6;
        [C1] cmpeq    C0:C1, vr0, vr3;
        [C1] cmplt    C0:C1, vr7, vr4, C0; //Ra=-2 かつ Rb' < 0xc0000000 か?
        [C1] aslww    vr7, vr7, 1;    //(Ra=-2 かつ Rb' < 0xc0000000)のとき
        [C1] cmplea   C2:C3, vr5, vr7, C1;
        [C2] lsr      vr8, vr0, 31;  //(Ra=-2 かつ Rb' < 0xc0000000) かつ
        [C2] lsr      vr9, vr1, 31;  // Ra' <= 2Rb' (ovf 判定) の場合
        [C2] xor      vr10, vr8, vr9; //オーバーフロー
        [C4] cmpeqn   C4:C5, vr10, 0, C2;
        [C4] mskgen   vr2, 30, 0;    //Rc = 0x7fffffff
        [C5] mskgen   vr2, 31, 31;   //Rc = 0x80000000
        [C3] extw     m0, vr11, vr0; //それ以外
        [C3] div      mh1, vr2, mh0, vr11, vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, float b)
{
    FIX16_1 c = b;
    return a/c;
}

```

【図 6 4】

```

inline FIX16_1 operator/(FIX16_1 a, double b)
{
    FIX16_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, float b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, double b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, float b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, double b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, float b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, double b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX16_1& FIX16_1::operator/=(FIX16_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(FIX16_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

```

【図 6 5】

```
inline volatile FIX16_1& FIX16_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(FIX32_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(FIX32_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}
```

【図 6 6】

```
}  
inline volatile FIX32_1& FIX32_1::operator/=(float b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
inline FIX32_1& FIX32_1::operator/=(double b)  
{  
    *this = *this / b;  
    return *this;  
}  
inline volatile FIX32_1& FIX32_1::operator/=(double b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
inline FIX16_2& FIX16_2::operator/=(FIX16_2 b)  
{  
    *this = *this / b;  
    return *this;  
}  
inline volatile FIX16_2& FIX16_2::operator/=(FIX16_2 b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
inline FIX16_2& FIX16_2::operator/=(int b)  
{  
    *this = *this / b;  
    return *this;  
}  
inline volatile FIX16_2& FIX16_2::operator/=(int b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
inline FIX16_2& FIX16_2::operator/=(float b)  
{  
    *this = *this / b;  
    return *this;  
}  
inline volatile FIX16_2& FIX16_2::operator/=(float b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
inline FIX16_2& FIX16_2::operator/=(double b)  
{  
    *this = *this / b;
```

【図 6 7】

```
    } return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(FIX32_2 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(FIX32_2 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

#pragma _enable_asm_end
```

【図 6 8】

```
#endif __AMPCC__  
#endif __FIXAMP__
```


【図 6 9】

```

/*****
*
* (C)Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*   funcamp.h
*   Version:
*   Release:
*   Date:    2002/6/19
*
*****/

/* 多重定義回避 */
#ifndef __FUNCAMP__
#define __FUNCAMP__

#include <stdlib.h>

#ifndef __AMPPCC__

long _abs(long);
long _max(long, long);
long _min(long, long);
long _adds(long, long);
long _subs(long, long);
int _bcnt1(long);
int _bseq0(long);
int _bseq1(long);
int _bseq(long);
int _log2(size_t size);
long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
void _clrm(long&, long&);
void _mul(long&, long&, long&, long, long);
void _mac(long&, long&, long&, long, long);
void _msu(long&, long&, long&, long, long);
void *_modulo_add(void *, int, int, size_t, void *);
void *_brev_add(void *, int, int, int, size_t, void *);

#else /* defined __AMPPCC__ */

#pragma _enable_asm_begin
#pragma _enable_inline_begin

long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
int _log2(size_t size);

static inline long
_abs(long data)
{
    long result;

    asm(vr0 = data){
        abs    vr1, vr0;
    }(result = vr1);

    return result;
}

```

【図 7 0】

```
static inline long
_max(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2){
        max vr2, vr1, vr0;
    }(result = vr2);

    return result;
}

static inline long
_min(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2){
        min vr2, vr1, vr0;
    }(result = vr2);

    return result;
}

static inline long
_adds(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2){
        adds vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline long
_subs(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2){
        subs vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline int
_bcmt1(long data)
{
    long result;

    asm(vr0 = data){
        bcmt1 vr1, vr0;
    }(result = vr1);

    return result;
}
```

【図 7 1】

```

static inline int
_bseq0(long data)
{
    long result;

    asm(vr0 = data){
        bseq0 vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int
_bseq1(long data)
{
    long result;

    asm(vr0 = data){
        bseq1 vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int
_bseq(long data)
{
    long result;

    asm(vr0 = data){
        bseq vr1, vr0;
    }(result = vr1);

    return result;
}

static inline void
_clrm(long &mh, long &m1)
{
    asm() {
        mul m0, vr1, vr0, 0;
    } (mh = mh0, m1 = m10);
}

static inline void
_mul(long &mh, long &m1, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b) {
        mul m0, vr2, vr0, vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void
_mac(long &mh, long &m1, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        mac m0, vr2, vr0, vr1, m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

```

【図 7 2】

```

}

static inline void
msu(long &mh, long &m1, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        msu m0, vr2, vr0, vr1, m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void *
modulo_add(void *addr, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = imm << tmp1;

    asm(vr0 = addr, vr2 = base, vr3 = tmp2, vr4 = tmp3) {
        mov    CFR0, vr3;
        add    vr6, vr0, vr4;
        addmsk vr7, vr2, vr6;
    } (p = vr7);

    return p;
}

static inline void *
brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3, tmp4;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = 16 - mask - tmp1;
    tmp4 = imm << tmp3;

    asm(vr0 = addr, vr1 = cnt, vr2 = base, vr3 = tmp2, vr4 = tmp3, vr5 = tmp4) {
        mov    CFR0, vr3;
        lsl    vr6, vr1, vr4;
        add    vr7, vr6, vr5;
        msbbrvh vr8, vr2, vr7;
    } (p = vr8);

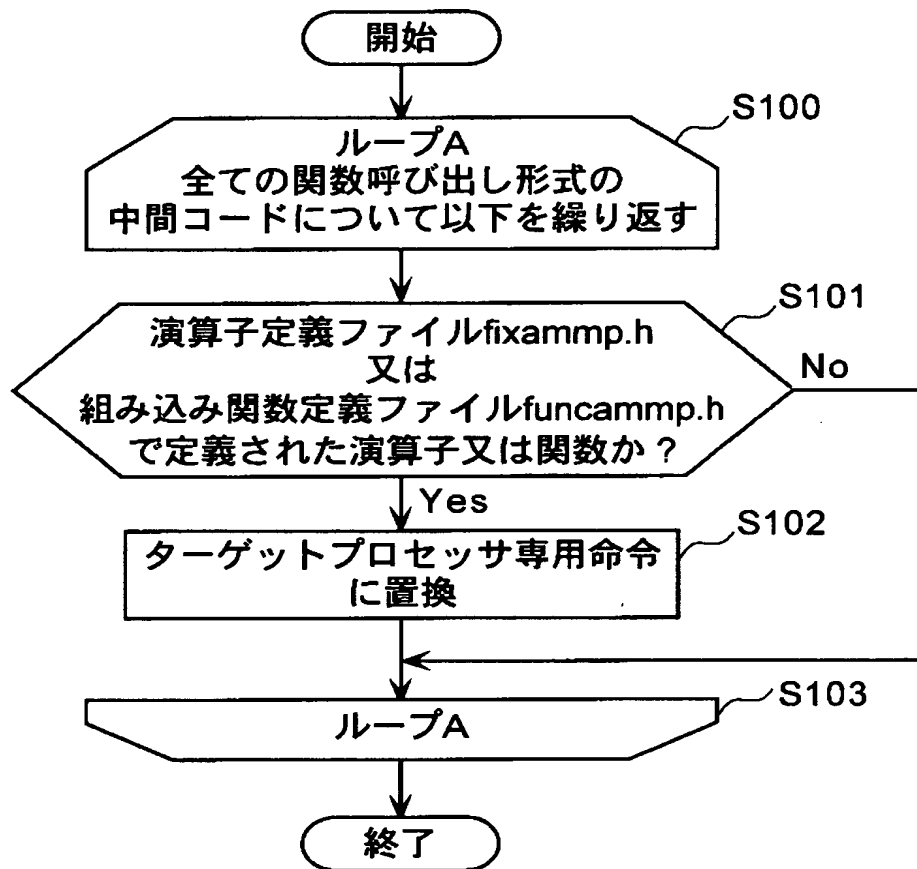
    return p;
}

#pragma _enable_inline_end
#pragma _enable_asm_end

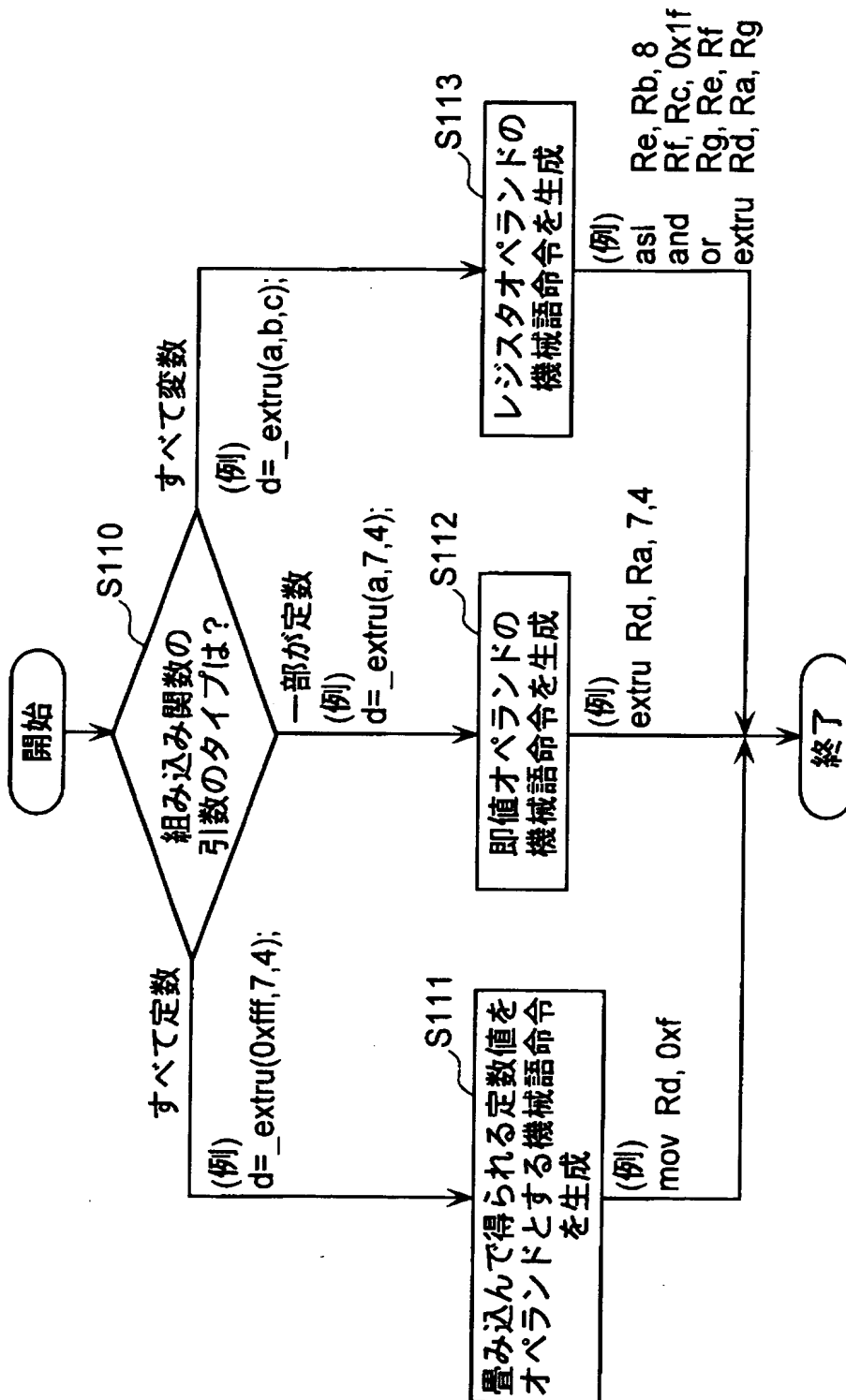
#endif /* __AMMPCC__ */
#endif /* __FUNCAMMP__ */

```

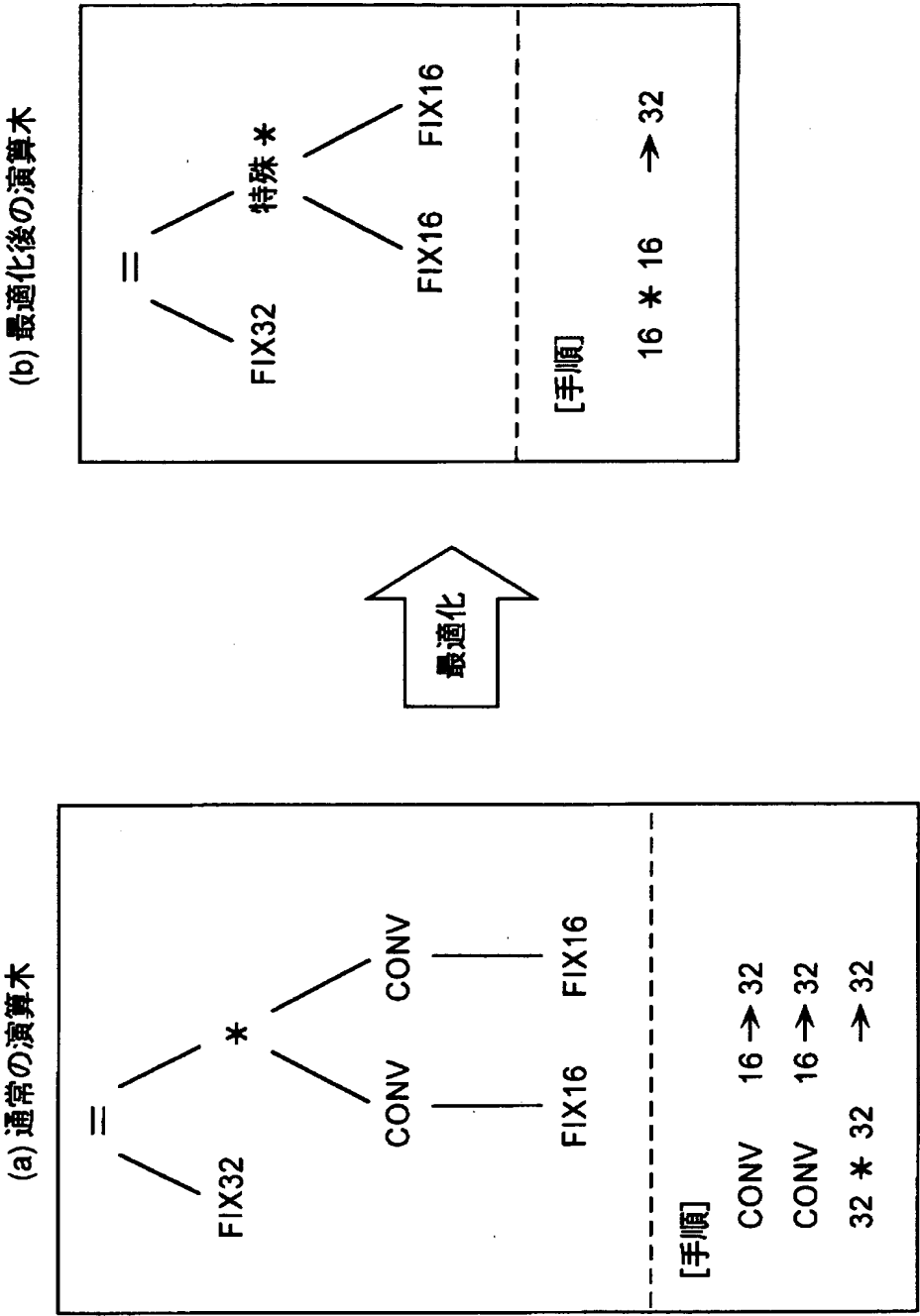
【図 7 3】



【図 7 4】



【図 7 5】



【図 76】

(a)

レイテンシ設定の例 1

```

<例>    wte と rde 間に 2 サイクル空ける
static inline int getbits(int a)
{
    int result ;
    asm (vr0 = a) {
        LATENCY L1, L2, 2 ;
        mov vr1, AVL_D_BASEADDR ;
L1:      wte C0:C1, (vr1, AVL_D_GETBITS), vr0 ;
L2:      rde C0:C1, vr2, (vr1, AVL_D_READPORT) ;
    } (result = vr2) ;
    return result ;
}

```

(b)

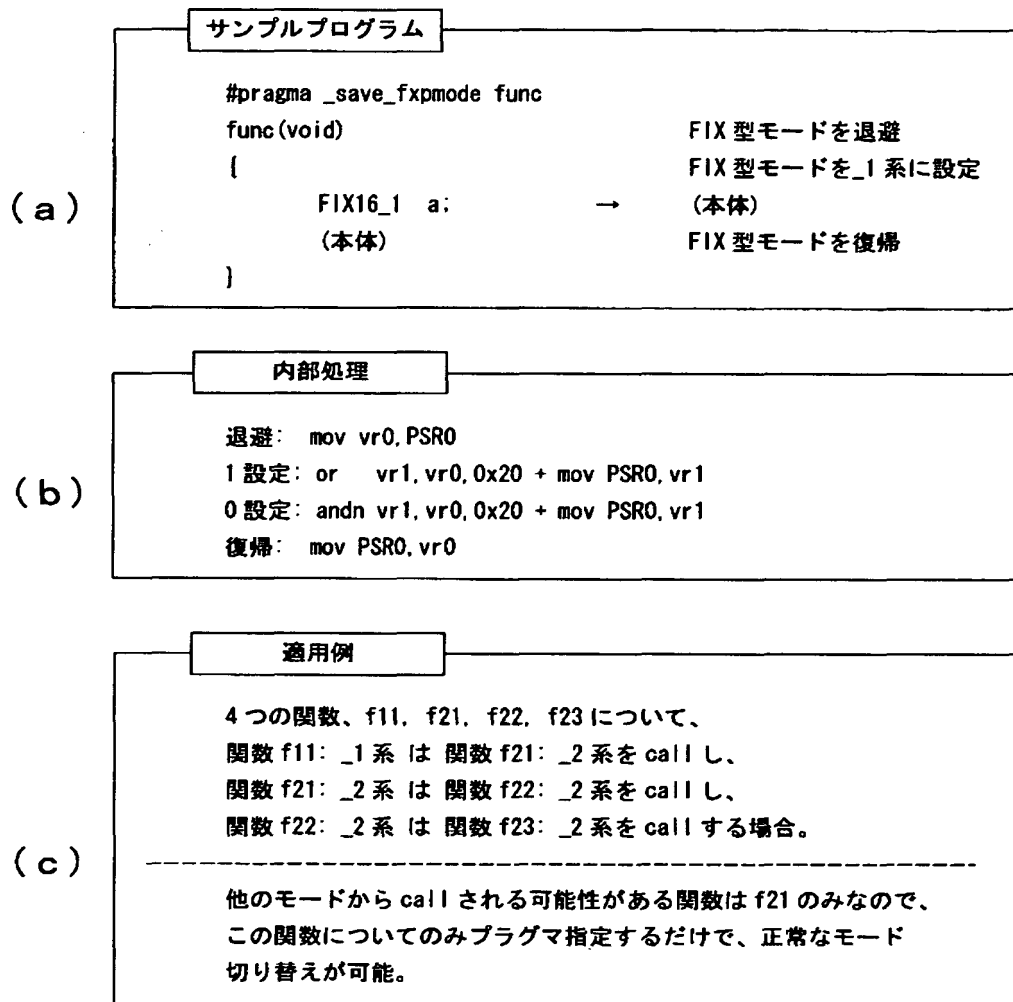
レイテンシ設定の例 2

```

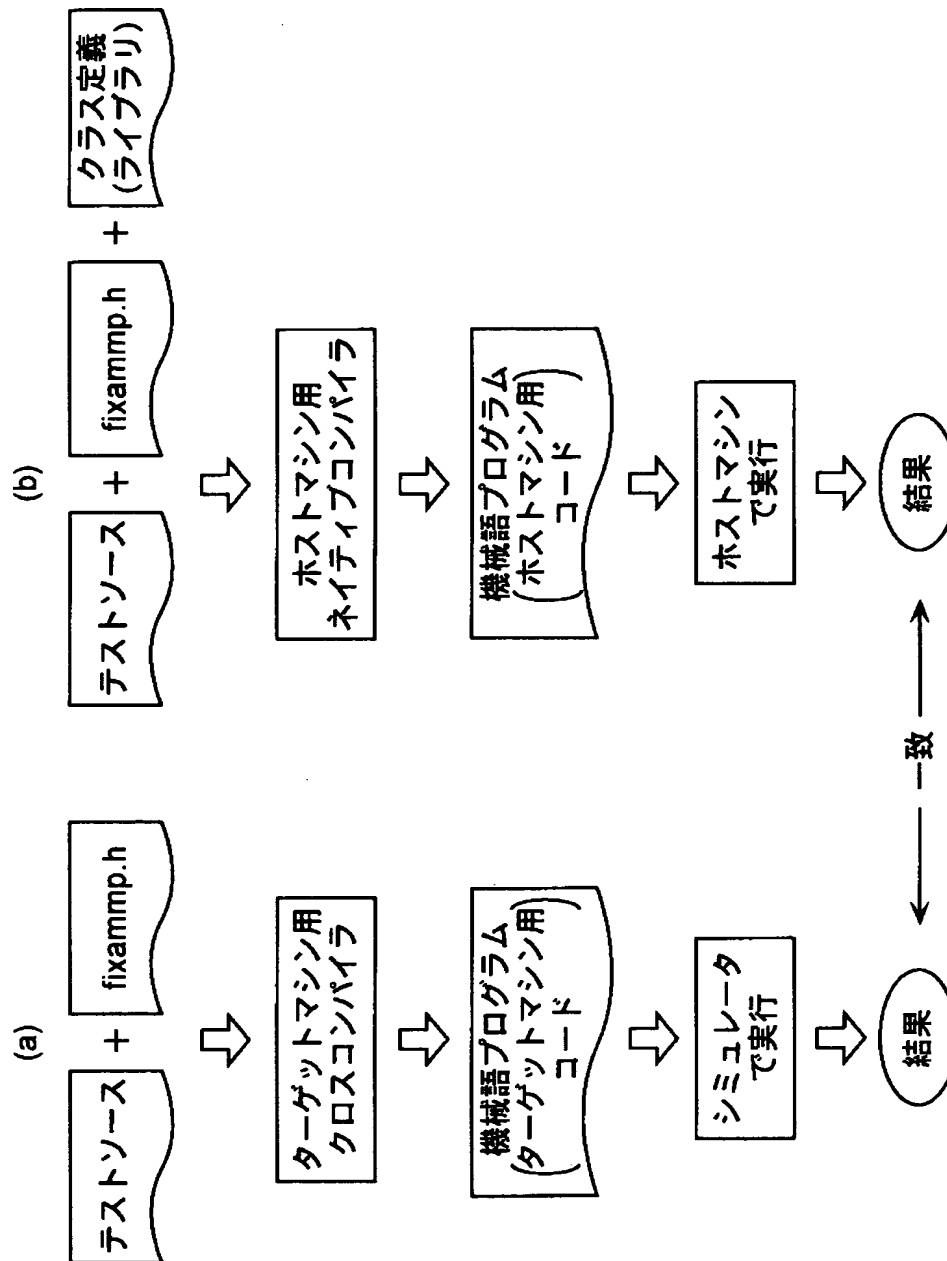
<例>    wte 後 次の拡張レジスタアクセスまで 2 サイクル空ける
static inline void skipbits(int a)
{
    asm (vr0 = a) {
        mov vr1, AVL_D_BASEADDR ;
        wte C0:C1, (vr1, AVL_D_SKIPBITS), vr0, LATENCY(2) ;
    } ;
}

```


【図 7 7】



【図 78】



【書類名】 要約書

【要約】

【課題】 プロセッサが備える高機能な専用命令を効率的に生成することができ、かつ、コンパイラ自身のバージョンアップを頻繁に繰り返すことなく、機能拡張等の改良を行っていくことが可能なコンパイラを提供する。

【解決手段】 ソースプログラム 1 0 1 にインクルードされる演算子定義ファイル 1 0 2 等と、ソースプログラム 1 0 1 を機械語プログラム 1 0 5 に翻訳するコンパイラとから構成され、演算子定義ファイル 1 0 2 にはクラス定義による各種固定小数点型の演算子の定義が含まれ、コンパイラ 1 0 0 は、中間コードを生成する中間コード生成部 1 2 1 と、演算子定義ファイル 1 0 2 で定義されたクラスを参照している中間コードを機械語命令に置き換える機械語命令置換部 1 2 2 と、置き換えられた機械語命令を含む中間コードを対象として最適化を行う最適化部 1 3 0 等を備える。

【選択図】 図 3 7

認定・付加情報

特許出願の番号	特願 2002-226682
受付番号	50201151947
書類名	特許願
担当官	第七担当上席 0096
作成日	平成14年 8月 5日

<認定情報・付加情報>

【提出日】 平成14年 8月 2日

出 願 人 履 歴 情 報

識別番号 [000005821]

1. 変更年月日 1990年 8月28日
[変更理由] 新規登録
住 所 大阪府門真市大字門真1006番地
氏 名 松下電器産業株式会社